NPS-OR-93-019

NAVAL POSTGRADUATE SCHOOL

Monterey, California

COMPUTER VISUALIZATION OF BATTLEFIELD TENETS

William G. Kemple
Harold J. Larson

December 1993

Approved for public release; distribution is unlimited.

Prepared for:
TRAC Monterey
Monterey, CA 93943-0692

FedDocs
D 208.14/2
NPS-OR-93-019

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

Rear Admiral T. A. Mercer
Superintendent

Harrison Shull
Provost

This report was prepared for and funded by TRAC Monterey.

Reproduction of all or part of this report is authorized.

This report was prepared by:

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1993	Technical

4. TITLE AND SUBTITLE
Computer Visualization of Battlefield Tenets

5. FUNDING NUMBERS
RKQHL

6. AUTHOR(S)
William G. Kemple and Harold J. Larson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER
NPS-OR-93-019

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
TRAC Monterey
P.O. Box 8692
Monterey, CA 93943-0692

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this report are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Battle Enhanced Analysis Methodologies (BEAM) project was designed to investigate the use of computer graphics in describing the performance of battalion-sized units in simulated combat. These descriptions were to be data-based and objective, providing useful critiques of actual performance according to standard Army doctrine. They would be natural candidates for use at the Army's Combat Training Centers.

The first year's effort demonstrated objective graphic displays that portray the destructive potential of direct fire weapons in the defense (described in [1,2,4,5]). These displays allow straightforward objective comparisons of different defensive alignments, and, from simulated battle runs, of defensive fire control strategies. These references also describe simple uncluttered displays that portray the movements and interactions of company (or higher) sized units throughout a battle.

This report describes further results of the BEAM project. The initial displays were specifically derived for direct fire weapons in the defense; a major development is the extension to displays for indirect fire weapons in the defensive. This allows separate and joint examination of the direct and indirect fire destruction potential, providing, among other things, objective measures of the synchronization and agility of a force, as well as indicators of its intelligence function.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Combat Power, Indirect Fire, Synchronization, Agility, Computer Graphics	66
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

COMPUTER VISUALIZATION OF BATTLEFIELD TENETS

William G. Kemple, Harold J. Larson
Naval Postgraduate School
Monterey, California 93943

Abstract

The Battle Enhanced Analysis Methodologies (BEAM) project was designed to investigate the use of computer graphics in describing the performance of battalion-sized units in simulated combat. These descriptions were to be data-based and objective, providing useful critiques of actual performance according to standard Army doctrine. They would be natural candidates for use at the Army's Combat Training Centers.

The first year's effort demonstrated objective graphic displays that portray the destructive potential of direct fire weapons in the defense (described in [1,2,4,5]). These displays allow straightforward objective comparisons of different defensive alignments, and, from simulated battle runs, of defensive fire control strategies. These references also describe simple uncluttered displays that portray the movements and interactions of company (or higher) sized units throughout a battle.

This report describes further results of the BEAM project. The initial displays were specifically derived for direct fire weapons in the defense; a major development is the extension to displays for indirect fire weapons in the defensive. This allows separate and joint examination of the direct and indirect fire destruction potential, providing, among other things, objective measures of the synchronization and agility of a force, as well as indicators of its intelligence function.

Introduction

To maximize combat readiness, the U. S. Army employs highly instrumented combat ranges for training troops under the most realistic possible conditions. Many of these ranges accommodate force-on-force battles, including simulated firings of weapons and of kills against the opponent; the

physical variables used in these simulated kills (times of events, locations of players) are then available for computer replays, including investigations of the effects of changes to some battle details.

A General Accounting Office report has criticized the Army for not making better use of objective observable measures in improving the training experience. The Battle Enhanced Analysis Methodology (BEAM) project has the important goal of using player and event time-location data to (objectively) visually portray the effectiveness of combat unit performance according to the standard battlefield tenets.

The ability of a force to inflict damage on an enemy at a given time varies from place to place on the battlefield. A surface whose height reflects the spatial distribution of this ability is useful for comparing performances of units which may employ different plans and/or tactics. Because such a surface is comparative in nature, its actual height for any given battlefield location is not of major importance. Its consistency in identifying locations at which the force has equal potential to do damage (equal heights) versus those locations where the potential is larger (greater heights) or smaller (lesser heights) is important, as is consistency across displays of alternate plans or tactics.

As previously reported in Battle Enhanced Analysis Methodologies (BEAM) reports (see [1,2,4,5]), this approach proves useful in describing the Destructive Potential (DP) of direct fire weapons in the defense and in examining the synchronization of these weapons. In these situations, the main factors under the commander's control that will cause differences in DP are the placements of the friendly weapons systems and the fire control measures employed. The height of the DP surface at any point on the battlefield is determined by the number and type of friendly weapon systems

which can engage the enemy at that point, the distances of these weapon systems from that point and the composition of the enemy force.

More specifically, the direct fire Destructive Potential $DP_{df}(x,y)$ at point (x,y) on the battlefield, for a battalion in the defense, is determined by the types and locations of defending weapon systems, their lines of sight, and associated probabilities of killing a target at the given point. Let index w identify a defending weapon system and let index t represent enemy target type; the weapon system may have two or more different armaments. For the given defensive position it then is possible to determine the points x,y on the battlefield which are within the effective ranges of the armaments and for which line of sight exists. Thus we can define Bernoulli (0,1) variables $L_{w,t}$ for each defending weapon system w and target type t; $L_{w,t}$ is 0 for all targets of type t at points x,y which are out of range of the armament or for which line of sight does not exist.

For each defending weapon system $p_{r,w,t}$ is the probability a target of type t at location x,y would be hit and killed by a round fired by armament w at range r from x,y. Granted armament w can fire $R_{w,t}$ rounds per minute, the expected number of kills to be made in one minute, by armament w against (instantaneously replaced) targets of type t at point x,y is $R_{w,t}L_{w,t}p_{r,w,t}$.

Granted an attacking force is composed of different target types, let f_t represent the fraction of the attacking force of type t. The Operational Lethality Index (*OLI*) for weapon type w against location x,y is defined by Lamont [3] as $OLI_w(x,y) = \sum_t f_t R_{w,t}L_{w,t}p_{r,w,t}$, the total number of expected kills to be made in one minute by this weapon system at location x,y. Finally, the direct fire Destructive Potential for the defending force, at point x,y, with weapon systems located at their given positions, is the sum of these OLI_w values:

$$DP_{df}(x,y) = \sum_{w} OLI_{w}(x,y) = \sum_{w}\sum_{t} f_t R_{w,t} L_{w,t} p_{r,w,t}.$$

This DP surface has units of *kills-per-minute*; it is not claimed to actually represent the expected number of kills which would be made at point x,y in any actual or simulated battle. Rather it provides easily interpreted *comparative* values for judging

- those areas in which the commander has chosen to concentrate his fire in selecting the locations for the defending weapons.
- which of several different defending force dispositions and/or fire control measures is better aligned with the commander's intent.
- the ability of the force, as deployed, to adjust its concentration of fires in response to the enemy's actions.

In addition to these Destructive Potential displays, simplified intuitive displays of the movements and interactions of company or higher sized units over the course of a battle have been described. These employ standard army symbology to identify units; the time trace of the locations of the units is easily visible, as is the dispersion within the units. These can provide clear uncluttered indications of the maneuver of several units through the course of the battle. They can also be used to provide indicators of the agility and intelligence functions of units (see the later discussion of synchronization, agility and intelligence).

All of these displays were constructed with data observed from units undergoing training at the National Training Center (NTC), Fort Irwin, California, using available computer hardware and software. Since no single available software package was capable of doing the necessary computing and graphic display, a mixture of platforms and programs was used. This made

the production of these displays very time consuming and limited their portability.

During the second year of this project, it was decided to investigate the use of different computer software, to make the production of displays faster and more straightforward, and to easily incorporate a graphical user interface. In addition, effort was concentrated on the production of graphic displays to show the effects of indirect fire, and the agility of forces. The results of these efforts are described in succeeding sections.

Software, user interface

The displays produced were derived from position-location data observed at the NTC, together with a digital terrain representation of the NTC. Different computer platforms were involved, as were numerous software packages which contributed various facets of the final displays. To unify and simplify this operation, the *TAE+* software package was investigated for applicability. This package has a well-developed graphical user interface, allowing one to choose and display any of many different objects; unfortunately the objects which it displays must be created with its own editor, which was incapable of handling the computational demands of lines of sight and destructive potential displays.

The software package PV-Wave was known to be capable of creating the graphics required, and was expected to soon have a graphics user interface which could easily interact with the BEAM displays. Since this user interface was not currently available, effort was concentrated on using the Wave control language to create graphics and to easily display them, doing away with the intermediate requirements of other platforms and software packages.

Several basic PV-Wave procedures have been derived, which are listed in the appendix. The first of these (*putpic.pro*) will read previously constructed displays (created in the original manner) into PV-Wave and display the surface. The resulting displays have been enhanced by adding battle graphics and a color bar to define the surface levels, both of which had previously been added with other software packages.

It had been agreed that destructive potential displays for indirect fire support would necessarily be dependent on the lines of sight of the observers who direct the fire. Thus, being able to determine the lines of sight of possible fire-direction observers is a part of the indirect fire combat potential. A PV-Wave procedure has been developed which will read the positions of the designated observers (at the desired time into the battle) from a player-location file, determine the lines of sight for all these positions (currently arbitrarily limited to a circle of radius 4000 meters), and then plot the accumulated lines-of-sight for these locations. This procedure is called *readlead.pro* and is also listed in the appendix; it effectively does away with the previously required intermediate steps of creating the *.srf* file required for *putpic.pro*. In addition, a method of capturing a graphic image previously displayed in PV-Wave into a file is described in the comments at the bottom of the listing for *putpic.pro*. Such a file can then be quickly recalled and redisplayed, avoiding the time-consuming line-of-sight determination required in initially building the display.

Indirect Fire

Direct fire weapons do not account for the total Destructive Potential for a defending battalion. Standard doctrine calls for additional support provided by indirect fire (from both mortars and artillery fire) as well as possible air

support. Commanders at all levels are responsible for integrating fire support into their plans (see reference [7], page 94, for more detail). The same basic approach is feasible for any and all of these; explicit attention has been given to the development of a DP surface for artillery fire, described and illustrated in this section.

The artillery fire DP surface will be derived in units of kills per minute, to be compatible with the direct fire surface; this allows addition of the two surfaces to see how well the two have been integrated in the overall plan, if desired. Review of doctrine and interviews with US Army and Marine Corps officers identified several variables that affect the use and effectiveness of artillery fire. Some of these officers have had command experience, and several had participated in training at the NTC on opposing forces, as observer controllers or with the training unit; some had Desert Storm experience. The experience of these officers was used to model the effects of key variables: the types of artillery weapons available together with the types of rounds employed, the numbers and locations of observers for controlling this fire, the locations of barriers and the target reference points (TRPs), the trafficability of the terrain and the enemy force mix.

It is assumed that artillery support is provided by an artillery battalion, which provides one or more Firing Units (FUs). The attacking force is assumed to be armored; thus the FUs are assumed to be $155mm$ and/or $8in$ tubes, since these are the only ones which are effective against this type of force. The Fire Direction Center (FDC) is assumed to know the locations of the FUs and their characteristics; it (and the FUs) knows the locations of the TRPs. Based on the officer experience already described, the probability of an artillery salvo killing a tank or Armored Personnel Carrier (APC), located at point x,y, is assumed to be $P_{max} = .9$, using Dual Purpose ICM ammunition. This is the

assumed maximum probability of a kill at each point under "ideal" conditions and information, for either type of FU; these ideal conditions may not exist at all x,y and P_{max} for targets at such points is reduced to account for this.

The "ideal" conditions which give the largest value for scoring a kill are

- An observer is sufficiently close to the point x,y, has LOS to that point, observes the target and calls for fire.
- The point x,y is within 2 kilometers of a TRP.
- The target is stationary at the given point.

(It is also assumed that the locations of the FUs allow them to range over the whole battlefield; if this does not hold, the discussion below is still appropriate by including one additional multiplicative factor with values 1 or 0, depending on whether the point x,y is within range of an FU.) The determination of the lines of sight is made by the procedure *test_LOS.pro*, listed in the appendix; the kernel of this procedure is the LOS algorithm in the JANUS(A) combat model. The procedure *find_range.pro* (see the appendix) was written to determine distances between points on the battlefield.

The probability of scoring a kill at x,y is degraded for any one or more of these conditions which do not hold; the way in which the probability of scoring a kill is degraded is to some degree arbitrary. Different weight functions describing the degradation in P_{max} were shown to selected officers; they were asked to choose which of these were most appropriate for the various factors, and to supply necessary parameter values for the functions. For example, the range from the fire control observer to the target should have an inverse effect on P_{max}; as this range increases, the probability of a salvo scoring a kill should decrease. A bell-shaped bisquare weight function

was recommended for this effect, described and pictured below. Separate procedures were written for each of the weight functions used (see *weight_bisquare.pro, weight_exponential.pro, weight_linear.pro* in the appendix); the parameter values needed are specified by procedure inputs. Both the shapes of the functions and their parameter values are easily changed in the indirect fire DP procedure (*arty.pro* listed in the appendix).

Four different functions have been employed in effecting the degradations to P_{max}; three of these were chosen from the officer discussions just described. The fourth was suggested later during a briefing of this material. The indirect fire displays to be presented below were derived using the functions to be described; the methodology employed is appropriate for any desired shapes and parameter values.

First, if one or more observers has line of sight to point x,y, then the *closest* observer is assumed to call fire on that point. The range from this closest observer to x,y is used to degrade the probability of scoring a kill. This is done using a bisquare weight function

$$f_1(r_1) = \begin{cases} .3 + .7\left(1 - r_1^2/36\right)^2, & \text{for } r_1 \le 6km \\ .3, & \text{for } r_1 > 6km, \end{cases}$$

pictured in Figure 1. With r_1 the range from the observer to x,y, $.3 \le f_1(r_1) \le 1$, with this smallest value occurring for all ranges of $6km$ or more. The probability P_{max} is multiplied by $f_1(r_1)$.

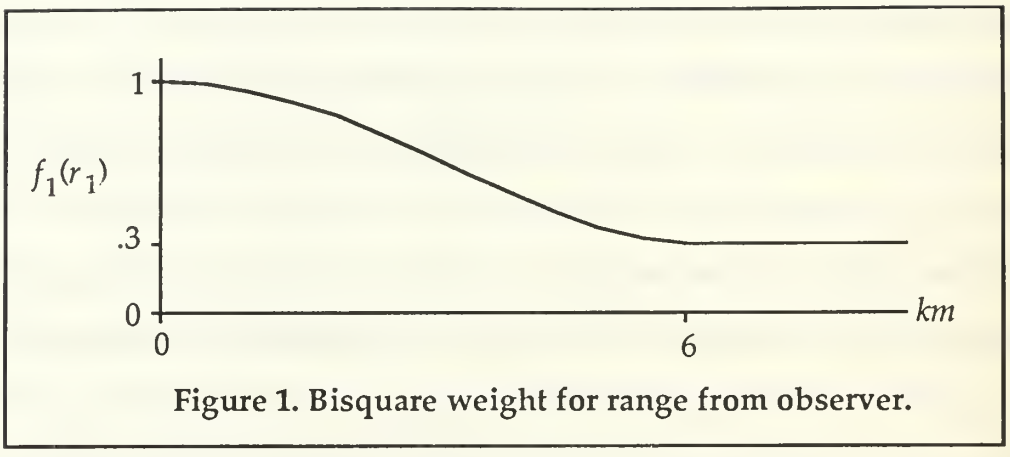

Figure 1. Bisquare weight for range from observer.

Second, the initial approach taken to account for whether or not the point x,y is close to a TRP was to use a "cookie cutter" function, pictured in Figure 2. The closer the desired firing point is to a TRP, the more likely a kill will be scored. This function is

$$f_2(r_2) = \begin{cases} 1, & \text{for } r_2 \le 2km \\ .65, & \text{for } r_2 > 2km, \end{cases}$$

used to adjust for the distance, r_2, from x,y to the closest useable TRP.

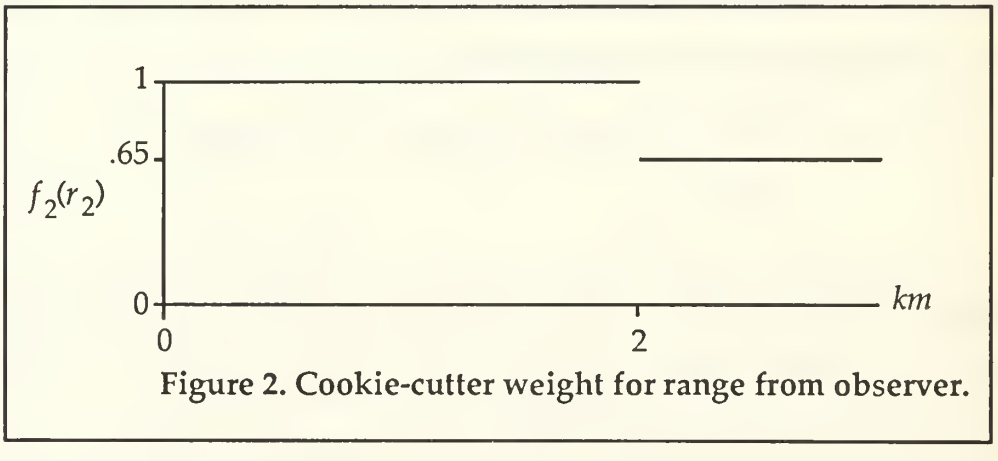

Figure 2. Cookie-cutter weight for range from observer.

This degradation caused by distance from the target to the closest TRP has also been modeled with a second function (suggested during a briefing of these results) which declines linearly from 1 to .65; this function is

$$f_2^*(r_2) = \begin{cases} 1 - .175r, & \text{for } r_2 \le 2km \\ .65, & \text{for } r_2 > 2km, \end{cases}$$

as pictured in Figure 3. Subsequent displays show the differences (and similarities) of results derived using these two functions. If x,y lies within $2km$ of a TRP, $P_{\max}$ is multiplied by $f_2(r_2)$ (for initial output displays) or by $f_2^*(r_2)$ for subsequently computed displays; in either case, if x,y is more than $2km$ from a TRP, $P_{\max}$ is multiplied by .65.

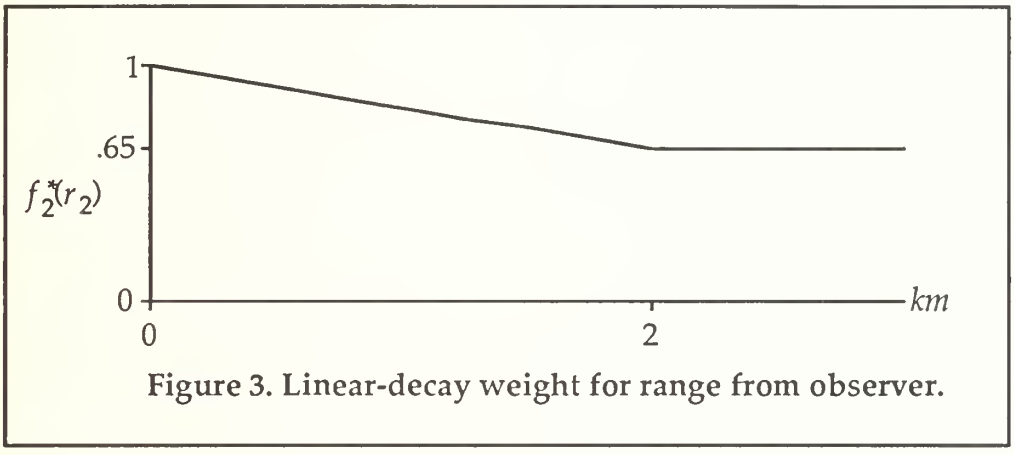

Figure 3. Linear-decay weight for range from observer.

Finally, the faster an enemy weapon system can travel, the less likely it is to be hit by artillery. This is accounted for by considering the trafficability at each point on the battlefield. Let s be the trafficability (expected speed of the target) at point x,y; if the target is stationary, then $s = 0$ and the probability of scoring a kill is not degraded. As s increases, the probability of scoring a kill is degraded by using the exponential decay function

$$f_3(s) = \begin{cases} .35 + .65\left(e^{-s/35} - e^{-1}\right)\Big/\left(1 - e^{-1}\right), & \text{for } s \le 35km/hr, \\ .35, & \text{for } s > 35km/hr, \end{cases}$$

pictured in Figure 4. For any target whose speed is $35km/hr$ or more, the multiplier is .35.

These degradation factors are the same for either type of FU employed. To summarize, for any given FU of type w the probability of scoring a kill at point x,y is the product $f_3(s)f_2(r_2)f_1(r_1)P_{\max}$. Again, as with direct fire, if FU w can fire R_w rounds per minute the expected number of kills scored per

minute at point x,y is the Operational Lethality Index $OLI_w(x,y) = R_w f_3(s) f_2(r_2) f_1(r_1) P_{max}$; granted two or more FUs may be employed the indirect fire Destructive Potential is the sum

$$DP_{if}(x,y) = \sum_w OLI_w(x,y).$$

Since both Destructive Potentials are in units of kills/minute, they may be added together to give the total Destructive Potential $DP(x,y) = DP_{df}(x,y) + DP_{if}(x,y)$.

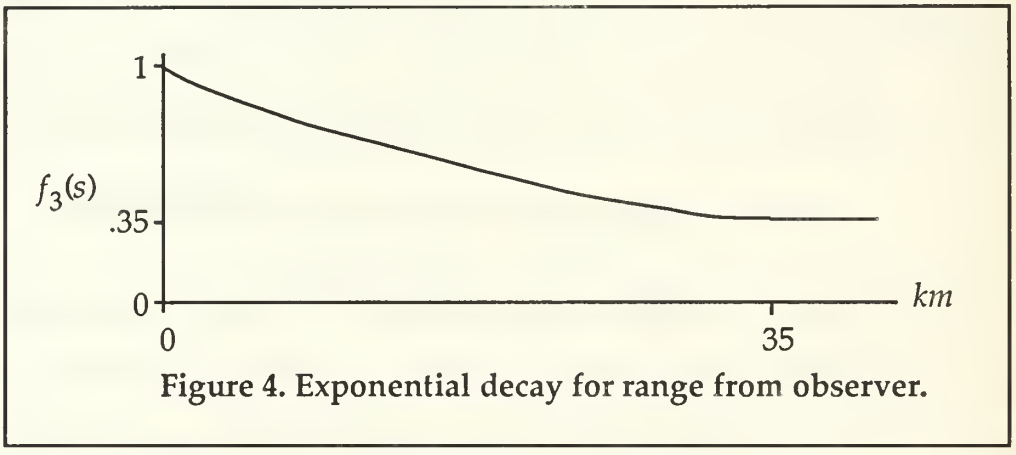

Figure 4. Exponential decay for range from observer.

The same NTC battle used for the direct fire Destructive Potential displays (see [5]) has been employed for $DP_{if}(x,y)$ displays. This featured an armor-heavy task force with a defense in sector mission; the task force commander intended to destroy the enemy in Engagement Areas SHARK and PIRANHA. The observed scenario was followed in initial locations of the defending force; twelve command leaders at various levels were designated as being the observers, who could then call for fire against designated targets. Assumed artillery support is provided by an artillery battalion, which supplies two FUs (155*mm* or 8*in* tubes).

To see their effect, a barrier is simulated in Engagement Area CUDA (this is done by reducing the trafficability with $f_3(s)$), and TRPs are placed at the corners of the engagement areas. To simulate the reductions in force caused

by casualties, half of the observers were chosen at random at time $T = 30$ minutes into the battle and were removed (killed), leaving six observers to call fire; again at time $T = 90$, half the remaining observers were removed, leaving three to call fire. At time $T = 90$ the barrier was assumed breached in two places and the defender's artillery rate of fire was cut in half; this was done to simulate the effects of tubes having become casualties, troop fatigue, logistic problems with maintaining supplies, *etc.*

A total of 6 $DP_{if}(x,y)$ displays are presented; these represent the resulting surface at times $T = 0$ (start of the battle), as well as $T = 30$ and $T = 90$. Figures 5, 7, and 9 use the original cookie-cutter function $f_2(r_2)$ to describe the degradation caused by the range between the point x,y and the closest TRP; Figures 6, 8, and 10 use the linear-decay function $f_2^*(r_2)$ for this same purpose. To sharpen the comparison of the cookie-cutter and linear-decay functions, the same observers were "randomly" removed at times 30 and 90, mentioned above, and the figures for the same times are displayed together.

The overall picture is the same in both three-figure sequences. First, the dramatic increase in DP_{if} caused by the barrier is evident throughout each sequence, as are the breaches in the barrier at $T = 90$. Next, moving forward in time, the loss of observers at $T = 30$ results in fewer areas with high DP_{if}, and the further loss of observers coupled with the reduction in firing rate at $T = 90$ result in very low DP_{if}, throughout the battlefield except at the barrier.

In addition, one can readily see the difference caused by the cookie-cutter versus the linear-decay function. In the first sequence of figures, at times $T = 0$ and $T = 30$, the cookie-cutter function causes large, homogeneous areas of high DP_{if}, making it difficult to observe the effect of distance from a TRP on DP_{if}. In the second sequence, using the linear-decay function, the DP_{if}, surface decreases smoothly away from the TRPs, highlighting their importance.

This approach for indirect fire DP can be easily modified in a number of ways. The degrading functions $f_1(\cdot), f_2(\cdot), f_3(\cdot)$ can be of any desired shape and value; the cutoff values where the functions remain constant can occur at other values and the constant achieved can be different. If other degrading factors are desirable they can easily be added in the same way. Other indirect fire DP measures (*e. g.* close air support) can be modelled and pictured in the same way. Procedures *artydraw.pro* and *artydraw2.pro* (see the appendix) were derived from *putpic.pro;* they can be used to simultaneously display several surfaces on the Wave screen, or to create postscript image files, that maintain the same color contour values for all figures.

Attack postures versus defense

As discussed, both $DP_{df}(x,y)$ and $DP_{if}(x,y)$ are defined for a defending force. The same basic concepts are appropriate for an attacking force. The destructive potentials are static measures, defined for a particular time epoch T. The indirect fire destructive potential $DP_{if}(x,y)$ for an attacking force can be defined in essentially the same manner as employed above. For an attacking force, the indirect fire Destructive Potential should profit from the targets being (essentially) static; it may suffer from the fact that the observer(s) calling for fire may be moving and that the targets may be in protected positions. Appropriate degradations for these effects can be easily incorporated into the indirect fire Destructive Potential.

The direct fire Destructive Potential for an attacking force can also be defined in a similar manner to that previously discussed. That is, at any given time T the locations of attacking vehicles are fixed (as are the locations of the defending force); lines of sight from these positions can be determined. If the attacking vehicles are capable of firing while moving (perhaps with degraded

probabilities of scoring hits and kills, as well as rates of fire) then again a surface representing expected kills per minute can usefully describe their destructive potential.

Indicators of Synchronization, Agility and Intelligence

Destructive Potential displays can be very useful in critiquing unit performance at the NTC. At the start of any given defensive battle, the DP surface should achieve its largest values at the areas of importance stressed by the commanders IPB; the color-coded display of the surface immediately indicates whether this has occurred or not. For persons well acquainted with the terrain over which the battle is to be fought, it is a relatively simple matter to find alternative defensive positions which result in a DP surface which achieves its maximum values at appropriate locations. This provides an objective critique of the commander's initial positioning of his forces in the defense.

The Destructive Potential surface for a force in the defense can be examined at any time in a simulated battle. Let T represent the time parameter for the battle, with $T = 0$ being the "starting time", and $DP_T(x,y)$ the height over x,y at time T. This surface depends only on the defensive locations, their lines of sight, firepower available and the initial enemy force mix; it is totally independent of actual enemy locations. In a simulated battle, it is possible to record the height of the DP surface over those points $x,y \in R_T$ on the battlefield which are actually occupied by enemy weapon systems, giving the *Theoretical Destructive Potential* at time T: $TDP_T = \Sigma_{x,y \in R_T} DP_T(x,y)$. With an attacking force, the set of occupied locations R_T will change with T as the vehicles move. The trace of this quantity over time proves useful for a number of things. If the defense is well synchronized and

agile, this quantity should be and remain close to its maximum possible as the battle progresses (T increases).

TDP_T, as just defined, will automatically decrease at any time T at which the defense kills an attacking weapon (since the set of occupied x,y positions R_T will decrease in size). If a particular attacker, located at position x',y', is killed at time T', let the height of the DP surface at that point (at that time) be $DP_{T'}(x',y')$. Define the *Realized Defensive Potential* at time T to be the sum of these heights where kills occurred, at times $T' \leq T$:

$$RDP_T = \sum_{(x',y')} DP_{T'}(x,y).$$

This quantity represents credit earned which should be given to the defense for kills already made by time T. Now define the *Applied Destructive Potential* to be $ADP_T = TDP_T + RDP_T$, the sum of the theoretical and realized destructive potentials.

ADP_T does not drop in value when the defense scores a kill; it does drop in value if the locations of the attacking systems move to places where the defender's Destructive Potential surface is lower. For a well-synchronized defense ADP_T should achieve and maintain a high value through time; this will occur only if the high values of the defender's DP surface correctly track the locations of the attacking weapon systems. Thus ADP_T provides an objective, numeric measure of the synchronization of the defending force.

Relative changes in ADP_T (over time) indicate the rate at which the Applied Destructive Potential is increasing or decreasing; the sign of the relative change indicates whether ADP_T is increasing (+) or decreasing (–). ADP_T is increasing whenever the defender is successful in placing higher values of DP over the enemy locations, and is decreasing when it is not successful in this goal. If the defense is agile, it should correctly anticipate the

enemy's actions, leading to ADP_T increasing (or at least not decreasing); if it is not agile, and does not accomplish this goal then ADP_T is decreasing (attack is agile). If ADP_T were actually measured continuously in time, then its derivative with respect to time T gives this measure of agility. Granted ADP_T will be computed only at specific times $T_0 = 0, T_1, T_2, \ldots$ this measure of agility can be computed at time T_j by

$$AG_{T_j} = \frac{ADP_{T_j} - ADP_{T_{j-1}}}{T_j - T_{j-1}}.$$

$AG_{T_j} \geq 0$ indicates the defense is agile (ADP is nondecreasing), while $AG_{T_j} < 0$ means the attacker is agile (ADP is decreasing).

This Applied Destructive Potential ADP_T can also be used in measuring the intelligence function of the defense. At time T, the defending force has an implied belief in the locations of the attacking enemy through the DP_T surface; that is, granted the defender is rational, the maximum value(s) of DP_T should correspond with the places the defender (through his intelligence) thinks the enemy is located (at time T). In any battle simulation, one also knows where the attacking weapon systems are in fact located. The worth or usefulness of the defender's intelligence is indicated by comparing these locations. If the defender's intelligence is perfect, the DP_T surface is high at the actual enemy locations and thus ADP_T is high. It is low if this is not the case. The previously discussed movement and maneuver displays can be used to get an objective measure of the "worth" of the defender's intelligence at time T.

Nelson [6] identified useful descriptors of the centroid (or middle) of a unit, as well as descriptors of its dispersion or geographic spread. For concreteness, let the centroid be determined by the median location and the dispersion by Nelson's convex hull (at time T). Define A_T as the volume

under DP_T over this convex hull (for the actual locations of the attackers at time T).

This centroid could then also be located at the maximum value of the DP_T surface, the location the defending force thinks is most likely for the enemy (at time T) and the convex hull can also be placed there. Now define E_T as the volume under DP_T over this "expected" convex hull (from the defender's point of view). The difference $INT_T = A_T - E_T$ will always be less than or equal to 0 and provides an objective measure of the quality of the intelligence of the defense (or of the quality of use of this intelligence). The closer INT_T is to 0, the better the intelligence of the defense; through time as the battle progresses, INT_T provides a trace of the intelligence usage of the defense.

References

[1] *Enhancing Tactical Direct Fire Synchronization Measures,* D. Dryer, W. Kemple, H. Larson, Proceedings of the 24th Symposium on the Interface: Computing Science and Statistics, 1992.

[2] *Toward Battlefield Visualization,* W. Kemple, H. Larson, S. Lawphongpanich, R. Lamont, M. Nelson, D. Dryer and J. Fernan, Proceedings of the Statistical Graphics Section, American Statistical Association, 1992.

[3] *Direct Fire Synchronization,* Major R. W. Lamont, USMC, MS Thesis in Operations Research, Naval Postgraduate School, September, 1992.

[4] *GRAPHICAL DISPLAYS OF SYNCHRONIZATION OF TACTICAL UNITS,* Harold J. Larson, William G. Kemple, Naval Postgraduate School, NPSOR-93-010, March, 1993.

[5] *Visualizing Synchronization of Tactical Units,* Harold J. Larson, William G. Kemple, Major David A. Dryer, USA, to appear in Mathematical and Computer Modelling.

[6] *Graphic methods for depicting combat unit locations, dispersion, and maneuver agility,* Captain M. S. Nelson, USA, MS Thesis in Operations Research, Naval Postgraduate School, September 1992.

[7] OPERATIONS, FM 100-5, Commander, TRADOC, Fort Monroe, VA 23651-5000, 5 May 1986.

PUTPIC.PRO:

```
; Procedure to replace Spyglass pictures
; Assumes line of sight has been determined, stored as before
; Reads data back in, puts up display
; Unless directed otherwise, scales colors from 0 to
; top of color scale for max observed.
; Syntax: putpic,no,top title (in quotes),filename to read (in quotes),
; color=color (to save the color bar).
; The first argument (no) is a fixed number for the window number
; (in case one wants to see two pictures simultaneously, odd for top
; even for bottom500).
; If one wants to scale the colors displayed, divide the phrase
; "bytscl(data)" by the desired factor (or multiply by a fraction).

PRO putpic,no,title,file,sntc,botlftxy,color,x,y

; data=fltarr(101,151)
data=fltarr(151,101)
status=dc_read_free(STRING(file),data)
;data=transpose(data)

color=intarr(2,251)
color(0,*)=indgen(251)
color(1,*)=indgen(251)

wave3_restore,'LOS.SAV' ; Brings back terrain heights and botlftxy

x=botlftxy(0)+.1*indgen(151)
y=botlftxy(1)+.1*indgen(101)
window,no MOD 2,xsize=776,ysize=440,xpos=50+50*(no MOD 2),$
      ypos=30+300*(no MOD 2)
shade_surf,sntc,x,y,az=0,ax=90,position=[.1,.1,.85,.95],$
      zaxis=-1,shade=bytscl(data),xtitle='KiloMeters',$
      ytitle='KiloMeters',/save,xrange=[45,60],yrange=[99.,109.],$
      ystyle=1

;xyouts,75,102,title,alignment=.5,size=1.3
      contour,sntc,x,y,nlevels=25,position=[.1,.1,.85,.95],/noerase,
      title=title,$ xrange=[45,60],yrange=[99,109],ystyle=1

x=45+.1*[50,0,18]
y=99+.1*[0,80,100]
oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ;phase line plot
x=45+.1*[80,150]
y=99+.1*[0,60]
oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ;phase line plot


x=45+.1*[72,82,92,98,101]
y=99+.1*[100,80,80,91,100]
oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ; piranha

x=45+.1*[30,35,65,68,40,30]
```

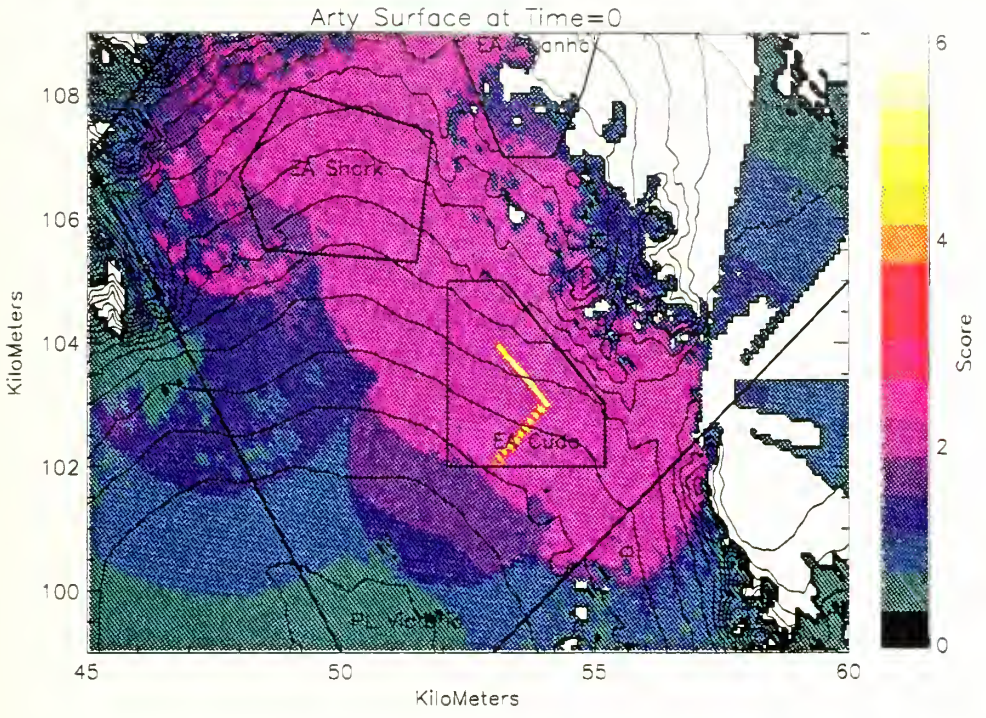

Figure 5. Indirect Fire Destructive Potential at T=0, Cookie-Cutter Weight Function.

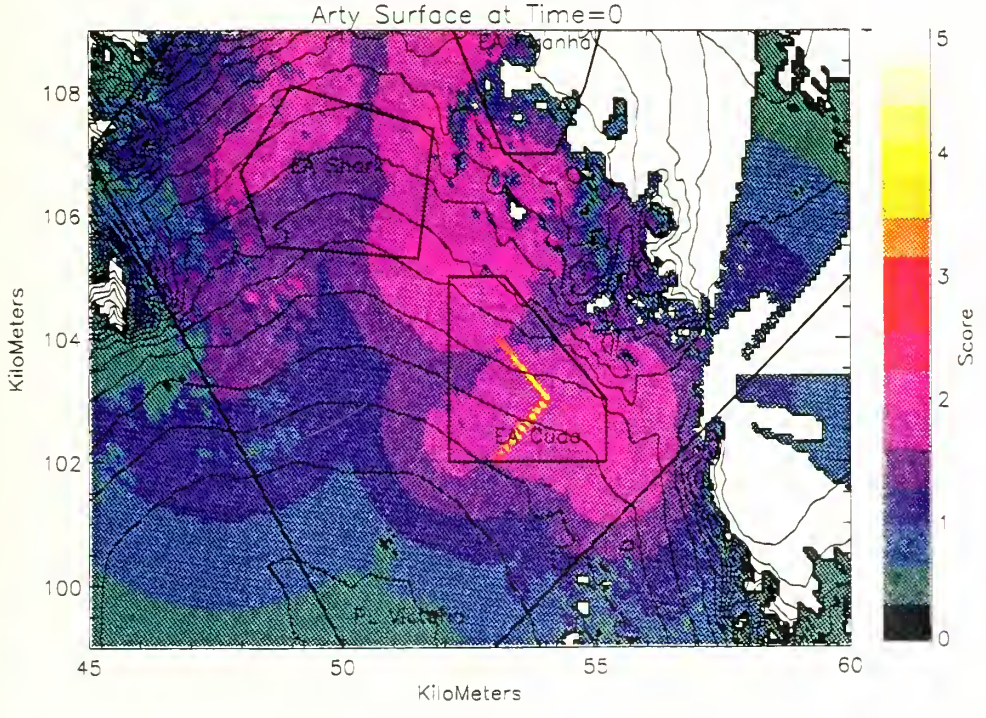

Figure 6. Indirect Fire Destructive Potential at T=0, Linear-Decay Weight Function.

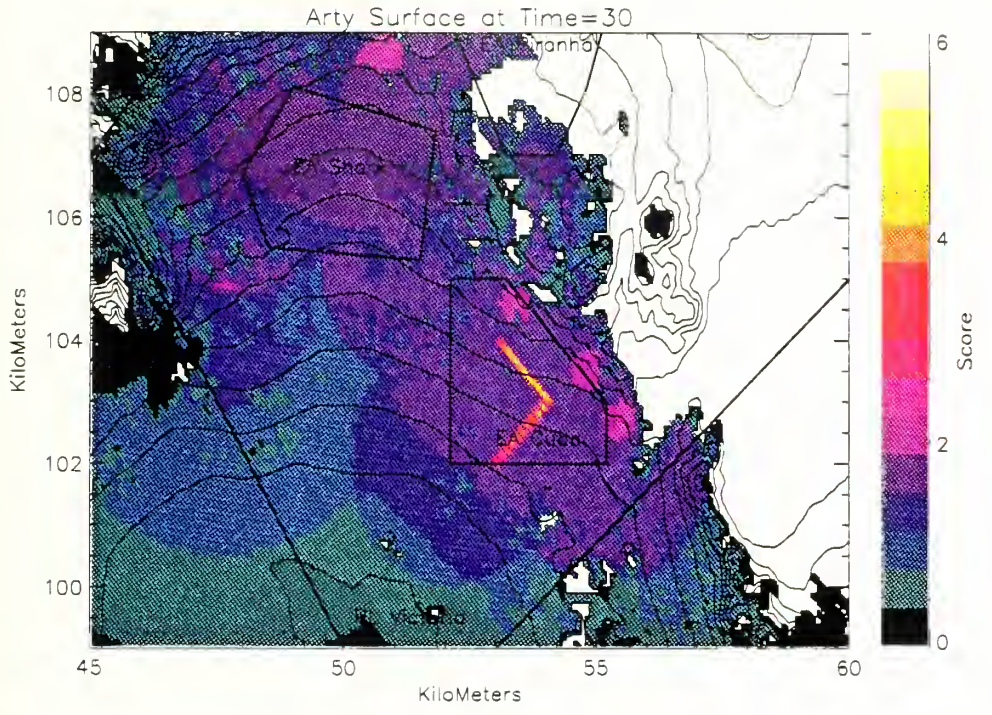

Figure 7. Indirect Fire Destructive Potential at T=30, Cookie-Cutter Weight Function.

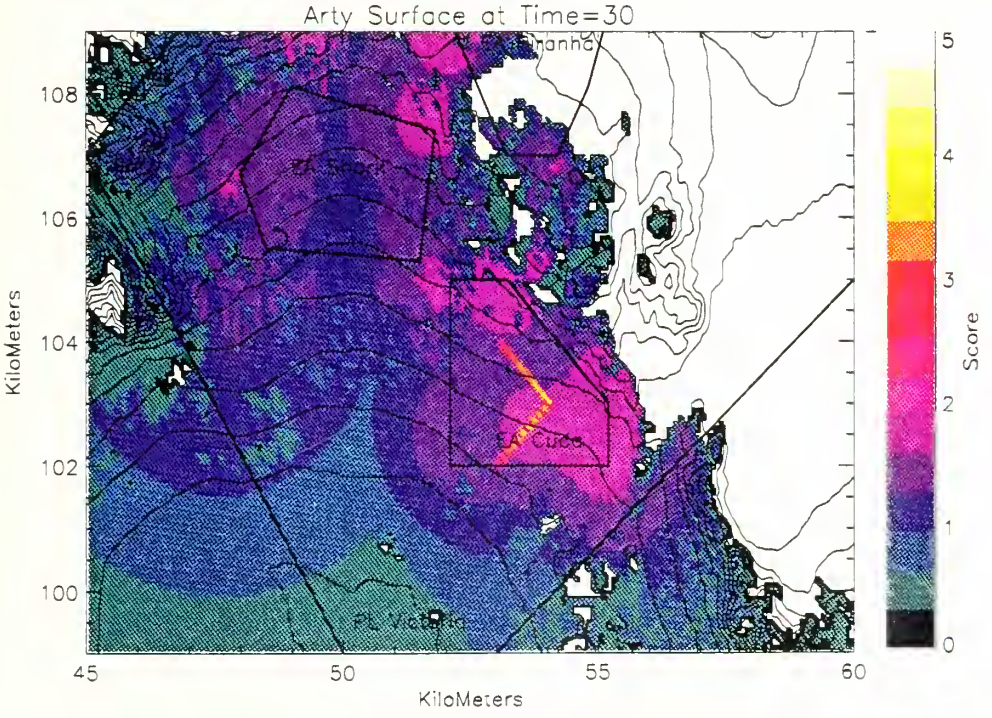

Figure 8. Indirect Fire Destructive Potential at T=30, Linear-Decay Weight Function.

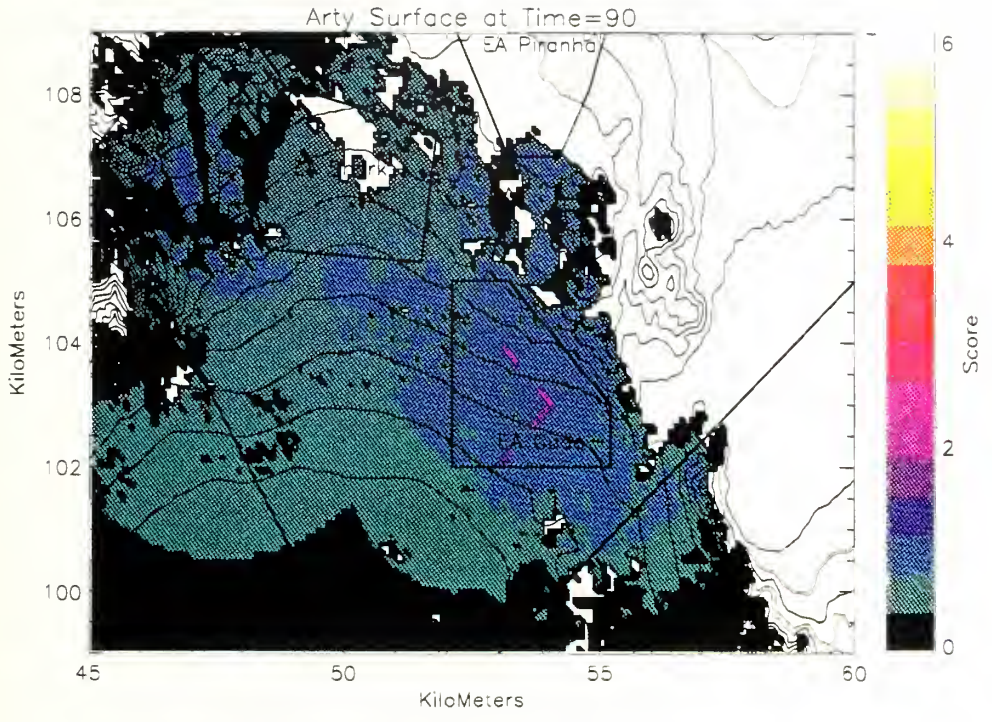

Figure 9. Indirect Fire Destructive Potential at T=90, Cookie-Cutter Weight Function.

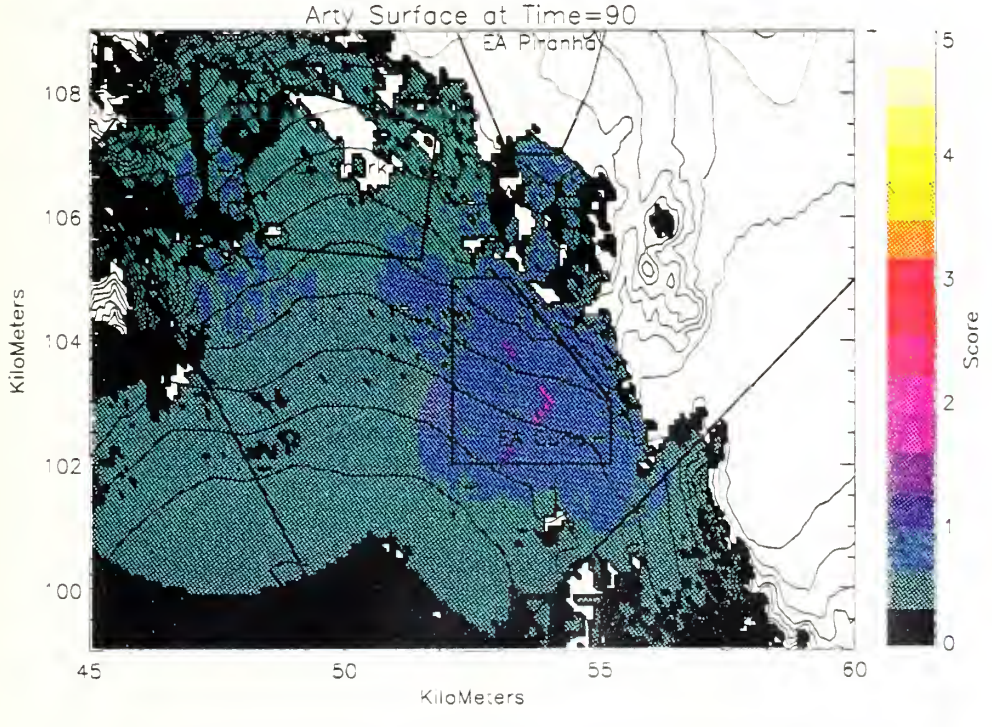

Figure 10. Indirect Fire Destructive Potential at T=90, Linear-Decay Weight Function.

```
y=99+.1*[77,65,63,84,91,77]
oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ; shark

x=45+.1*[71,71,102,102,82,71]
y=99+.1*[60,30,30,40,60,60]
oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ; cuda

xyouts,50.2,99.4,'PL Victoria'

xyouts,53,102.3,'EA Cuda'

xyouts,49,106.7,'EA Shark'

xyouts,52.7,108.7,'EA Piranha'



shade_surf,color,az=0,ax=90,position=[.88,.1,.93,.95],shades=color,$
          xstyle=4,ystyle=4,/noerase

axis,position=[.88,.1,.93,.95],xstyle=4,yaxis=1,ytitle='Score',/noerase



END
; A displayed image can be stored in a Wave variable, and then quickly
; redisplayed.
; For a window in a given position, the full picture goes from (0,0) to
; (x,y) = (max X, max Y) - 1. With the image displayed, it is put into
; variable image by
;                              image=tvrd(0,0,x,y).
; With a window located at the same place, the image is displayed with
;                               tv,image
;
; Window 0 in putpic is restored with window,no,xpos=50,ypos=30,$
;   xsize=776,ysize=440
; If image captured the previous display, it is then redisplayed with
; above command: tv,image
;
; To save such an image to file, use the save command:
;     save,image,xpos,ypos,xsize,ysize,filename='name',/verbose
;
; It can then be recovered later by restore,'name',/verbose
```

READLEAD.PRO:

```
PRO readlead,leaders,x,y,leader=leader,SNTC,BOTLFTXY

; The vector leader specifies the indices of the positions identifying the
; leaders. This is set up to specifically read from the initial position
; file gplt.dat; the number of leaders is 34 and there are 279 positions in
; this file for time 01:58:10. The indices of the leaders originally given
; by Jude can be restored by the command
;             restore,'leader.dat',/verbose
;
; This brings back the single vector named L which contains the indices of
; the specified individuals. The image currently generated simply sums the
; numbers of individuals who have line of sight to the given point, using
; Dave Dryer's LOS algorithm. This image can be quickly retrieved with
;             restore,'leader.img',/verbose
; and then viewed with the procedure given at the end of the putpic.pro
; file.



openr,1,'/home2/ntc/jcf/dad/ingres/gplt.dat'

y=intarr(34) ; size determined by leader, array of identifiers
x=strarr(279) ; size determined by number of records desired


readf,1,x  ; reads in all data for time 01:58:10

close,1

color=intarr(2,251)  ; Used to put up the color bar on the right.
color(0,*)=indgen(251)
color(1,*)=indgen(251)

; Now go with unit numbers in fifth column, should be unique.

for i=0,33 do y(i)=max((float(strmid(x,29,4))EQ leader(i))*indgen(279))

; endfor

leaders=x(y(where(y GT 0)))  ; identifies possible locations for callers

XARR=float(strmid(leaders,36,6))/1000 ; x-coordinates in correct system
print,'size of XARR: ',size(XARR)
YARR=float(strmid(leaders,43,6))/1000 ; y-coordinates
YARR(where(YARR LT 70))=YARR(where(YARR LT 70))+100 ; corrected
  y-coordinates
HFARR=intarr(34)
HFARR(*)=0  ; All holdfires=0
RNGARR=intarr(34)
RNGARR(*)=4000  ; All sighting ranges=4000


wave3_restore,filename='LOS.SAV'
```

```
; The following code is lifted from LOSNEWMOD.PRO

MAPXY = SNTC
MAPARAM = SIZE(MAPXY)
XRNGMAP = FIX(MAPARAM(1))
YRNGMAP = FIX(MAPARAM(2))


H = 1
L = 0
N = N_Elements(XARR)

LABELX:
LOSURF = INTARR(XRNGMAP,YRNGMAP)
LABELY:
PRINT,L+1, ' OUT OF ',N
IF (L EQ N-1) THEN GOTO, LABELZ

LABELZ:
HF = FIX(HFARR(L))
IF (HF EQ 1) THEN GOTO, SKPWPN
IF ((XARR(L) LT 45) OR (XARR(L) GT 60) OR (YARR(L) LT 99) OR (YARR(L) GT
109)) THEN GOTO, SKPWPN XS = (XARR(L) - BOTLFTXY(0)) * 10
YS = (YARR(L) - BOTLFTXY(1)) * 10
RNG = RNGARR(L)
IXS = FIX(XS)
IYS = FIX(YS)

GRIDS = INTARR(2,6000)
DISTARR = FLTARR(1,6000)
NUMGRIDS = 1
RNGDIST = RNG / 100
BXG = FLOAT(IXS)
BYG = FLOAT(IYS)
INCXARR = [1., -1., -1., 1.]
INCYARR = [1., 1., -1., -1.]
INITSUBX = [0., 1., 1., 0.]
INITSUBY = [0., 0., 1., 1.]
GRIDS(*,0) = [BXG, BYG]
FOR I = 0, 3 DO BEGIN
XG = BXG - INITSUBX(I)
YG = BYG - INITSUBY(I)
LABELA: DIST = SQRT((XG-XS)^2 + (YG-YS)^2)

IF (DIST LT RNGDIST) THEN BEGIN
  IXG = FIX(XG)
  IYG = FIX(YG)
  GRIDS(*,NUMGRIDS) = [IXG, IYG]
  DISTARR(*,NUMGRIDS) = [DIST]

  NUMGRIDS = NUMGRIDS + 1
      XG = XG + INCXARR(I)
      GOTO, LABELA

  ENDIF ELSE BEGIN
      YG = YG + INCYARR(I)
      XG = BXG - INITSUBX(I)
```

```
      DIST = SQRT((XG-XS)^2 + (YG-YS)^2)
      IF (DIST GT RNGDIST) THEN GOTO, LABELB
      GOTO, LABELA
ENDELSE

LABELB:
ENDFOR

GRIDS = GRIDS(*,1:NUMGRIDS)
DISTARR = DISTARR(*,1:NUMGRIDS)

FOR J = 0, NUMGRIDS-1 DO BEGIN

XT = FLOAT(GRIDS(0, J))
YT = FLOAT(GRIDS(1, J))
RANGE = DISTARR(0, J)
IXT = FIX(XT)
IYT = FIX(YT)
IELEV = FLTARR(500)
IF (IXT LT 0) OR (IXT GT XRNGMAP-1) OR (IYT LT 0) OR (IYT GT YRNGMAP-1)
THEN GOTO, LABEL5

IDX = ABS(IXT - IXS)
IDY = ABS(IYT - IYS)
IF ((IDX EQ 0) AND (IDY EQ 0)) THEN GOTO, LABEL4

;PRINT, 'IDX, IXY          =', IDX, IDY
IP = 1
IF (IDY GT IDX) THEN GOTO, LABEL1

Y=YS
DY = (YT - YS) / FLOAT(IDX)
INC = 1
IF (XT LT XS) THEN INC = -1

FOR  IX = IXS, IXT, INC DO BEGIN
   IY = FIX(Y)

 ZZ = MAPXY(IX,IY)
   IELEV(IP) = ZZ

 Y = Y + DY
   IP = IP + 1
ENDFOR
GOTO, LABEL2

LABEL1:  ;--STEP IN Y--
X = XS
DX = (XT - XS) / FLOAT(IDY)
INC = 1
IF (YT LT YS) THEN INC = -1

FOR IY = IYS, IYT, INC DO BEGIN
;             (CHANGED FROM: FOR IY = IYS-1, IYT, INC DO BEGIN)
   IX = FIX(X)
ZZ = MAPXY(IX,IY)
   IELEV(IP) = ZZ
```

```
;PRINT, 'IELEV               =', IELEV(IP)

  X = X + DX
  IP = IP +1
ENDFOR

LABEL2:  ITRGT = IP - 1

ZO = 6
ZOBS = ZO + IELEV(1)
PLOS = 0
ZT = IELEV(ITRGT) + 2
ISTOP = ITRGT - 1
DZ = (ZT-ZOBS) / FLOAT(ISTOP)

Z = ZOBS
FOR I = 2, ISTOP  DO BEGIN
  Z = Z + DZ
  ZZ = IELEV(I)
IF (Z LT ZZ) THEN GOTO, LABEL3        ;--NO LOS
ENDFOR

LABEL4:  PLOS = 1
LOSURF(IXT,IYT)=LOSURF(IXT,IYT) + 30


GOTO, LABEL5




LABEL3:




LABEL5:
ENDFOR


SKPWPN:
IF (L EQ N-1) THEN BEGIN
WINDOW, 6,xsize=776,ysize=440,xpos=50,ypos=30
x=botlftxy(0)+.1*indgen(151)
y=botlftxy(1)+.1*indgen(101)

 SHADE_SURF,MAPXY,x,y, AZ=0, AX=90, ZAXIS = -1, SHADE=BYTSCL(LOSURF), $
    position=[.1,.1,.85,.95],xtitle='KiloMeters',$
    ytitle='KiloMeters',/save,xrange=[45,60],yrange=[99.,109.],$
    ystyle=1

  title='LOS for leaders'
   CONTOUR, MAPXY,x,y, /T3D, NLEVELS=29,$
      position=[.1,.1,.85,.95],/noerase,title=title,$
      xrange=[45,60],yrange=[99,109],ystyle=1
```

```
   x=45+.1*[50,0,18]
   y=99+.1*[0,80,100]
   oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ;phase line plot
   x=45+.1*[80,150]
   y=99+.1*[0,60]
   oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ;phase line plot


   x=45+.1*[72,82,92,98,101]
   y=99+.1*[100,80,80,91,100]
   oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ; piranha

   x=45+.1*[30,35,65,68,40,30]
   y=99+.1*[77,65,63,84,91,77]
   oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ; shark

   x=45+.1*[71,71,102,102,82,71]
   y=99+.1*[60,30,30,40,60,60]
   oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ; cuda

xyouts,50.2,99.4,'PL Victoria'

xyouts,53,102.3,'EA Cuda'

xyouts,49,106.7,'EA Shark'

xyouts,52.7,108.7,'EA Piranha'


shade_surf,color,az=0,ax=90,position=[.88,.1,.93,.95],shades=color,$
           xstyle=4,ystyle=4,/noerase


axis,position=[.88,.1,.93,.95],xstyle=4,yaxis=1,ytitle='Score',/noerase


PRINT,'END'
  GOTO, LABELZZ
ENDIF
H = H + 1
  L = L + 1

  GOTO, LABELX

  L = L + 1
  GOTO, LABELY

LABELZZ:

;  RETURN apparently not needed
END
```

TEST_LOS.PRO:

```
FUNCTION test_LOS, in_xs, in_ys, height_s, in_xt, in_yt, height_t

    COMMON BILL, Isntc, BOTLFTXY
; ********************************************************************
; Bill Kemple                                    last update  19 july 1993
;
; function to test line of sight (LOS) between a sensor at
; (xs,ys), height_s feet above the terrain and a target at
; (xt,yt), height_t feet above the terrain.
;
;;
;; ********************************************************************
;; there are several print statements included to help in debugging
;;    if debug is set to 1, they will print, otherwise they will not
;;
;; debug can be set in the workspace or main program by commenting
;;    the next line out, or here by using it
;; ------------------------------------------------------------------
     debug = 0
;; ------------------------------------------------------------------
;;   modified 12 Jun 93 to add target height (for trps)
;;      need to carry this through
;;
;   --FROM JANUS(A) 2.0 FASTUP.FOR
;   Isntc -two dimensional array of grid elevation cell data (nearest foot)
;   elev  -array of terrain elevation data (meters) along
;            the los projection from sensor to target
;
;   20 june 93 changed input files to give 3 digit (1 unit = 100m) coords
;;
;   our cells are in hundreds of meters, and the indices start
;        at (0,0), so subtract the smallest grid value.
;
;   we want the lower left corners of the cells that our points are in.
;     the entire cell is assumed to have the same elevation.
;        converting the coordinates to integer truncates them.
;
; ------------------------------------------------------------------
   xs = in_xs - 450.
   ys = in_ys - 990.

   xt = in_xt - 450.
   yt = in_yt - 990.

   Ixs = FIX(xs)
   Iys = FIX(ys)

   Ixt = FIX(xt)
   Iyt = FIX(yt)

   elev = FLTARR(500)    ; array to hold the elevation of cells between
                                ; our points

; ********************************************************************
;; for bounds check, need min & max values -- for now, hard wire them
```

```
;
; ---------------------------------------------------------------------
  IF (Ixs LT 0) OR (Ixs GT 150) OR (Iys LT 0) OR (Iys GT 100) $
    THEN BEGIN
      PRINT, 'Bad xs or ys  ', xs, ys
      RETURN, LOS_FLAG = 0
  ENDIF

  IF (Ixt LT 0) OR (Ixt GT 150) OR (Iyt LT 0) OR (Iyt GT 100) $
    THEN BEGIN
      PRINT, 'Bad xt or yt  ', xt, yt
      RETURN, LOS_FLAG = 0
  ENDIF
; ---------------------------------------------------------------------
;; for debugging
  IF debug EQ 1 THEN BEGIN
    PRINT, 'xs, ys   =', xs, ys
    PRINT, 'xt, yt   =', xt, yt
  ENDIF
; ---------------------------------------------------------------------
; *********************************************************************
; if the points are in the same cell, LOS exists, and we need go
;   no farther. if not, start the big loop.
;
; calculate the delta_x and delta_y values and see which is larger.
;   full unit moves are made in the direction of the larger, and
;      fractional moves for the smaller.
;
; ---------------------------------------------------------------------
  IF Ixs EQ Ixt AND Iys EQ Iyt THEN BEGIN
    RETURN, LOS_FLAG = 1
  ENDIF

    Idelta_x = ABS(Ixt - Ixs)
    Idelta_y = ABS(Iyt - Iys)

; ---------------------------------------------------------------------
;; for debugging
  IF debug EQ 1 THEN BEGIN
    PRINT, 'FASTUP CALCULATION:'
    PRINT, 'SENSOR xs, ys      =', xs, ys
    PRINT, 'TARGET xt, yt      =', xt, yt
    PRINT, 'SENSOR Ixs, Iys    =', Ixs, Iys
    PRINT, 'TARGET Ixt, Iyt    =', Ixt, Iyt
    PRINT, ' '
    PRINT, 'Idelta_x, = Idelta_y =', Idelta_x, Idelta_y
  ENDIF
; ---------------------------------------------------------------------
; *********************************************************************
; start the big loop
;
; ---------------------------------------------------------------------

  Istep = 0
  IF (Idelta_y LT Idelta_x) THEN BEGIN    ; --STEP IN X--
; ---------------------------------------------------------------------
;; for debugging
  IF debug EQ 1 THEN BEGIN
```

```
      PRINT, ' '
      PRINT, '------------STEP IN X-----------'
      PRINT, ' '
    ENDIF
;   ------------------------------------------------------------------------
;   ************************************************************************
;   whole moves will be made in the x direction, see if the target is
;     left (xt < xs) or right (xs < xt) from the sensor
;
;   next, need the fraction to move in the y direction for each whole
;     increment in the x direction
;
;   these increments will be accumulated and only used when they add up
;     to whole steps
;   ------------------------------------------------------------------------
      IF (xt LT xs) THEN BEGIN
        increment = -1
      ENDIF ELSE BEGIN
        increment = 1
      ENDELSE

      y_increment = (yt - ys) / FLOAT(Idelta_x)

      y_accumulator = ys
;   ------------------------------------------------------------------------
;   ************************************************************************
;   find the elevation for every cell between the sensor and the target
;     and store them in elev for the LOS test
;   ------------------------------------------------------------------------
      FOR  Ix_cell = Ixs, Ixt, increment DO BEGIN
        Iy_cell = FIX(y_accumulator)

;   ------------------------------------------------------------------------
;;  for debugging
    IF debug EQ 1 THEN BEGIN
    PRINT, 'Istep, Ix_cell, Iy_cell         =', Istep, Ix_cell, Iy_cell
    ENDIF
;   ------------------------------------------------------------------------

        elev(Istep) = Isntc(Ix_cell,Iy_cell)

;   ------------------------------------------------------------------------
;;  for debugging
    IF debug EQ 1 THEN BEGIN
      PRINT, 'elev              =', elev(Istep)
    ENDIF
;   ------------------------------------------------------------------------
        y_accumulator = y_accumulator + y_increment
        Istep = Istep + 1
      ENDFOR

    ENDIF ELSE BEGIN  ; end of step in x, start step in y
;   ************************************************************************
;
;   -- STEP IN Y--
;   ------------------------------------------------------------------------
;;  for debugging
    IF debug EQ 1 THEN BEGIN
```

```
      PRINT, ' '
      PRINT, '--------------STEP IN Y------------'
      PRINT, ' '
    ENDIF
;  ---------------------------------------------------------------------
;*********************************************************************
;  whole moves will be made in the y direction, see if the target is
;    down (yt < ys) or up (ys < yt) from the sensor
;
;  next, need the fraction to move in the x direction for each whole
;    increment in the y direction
;
;  these increments will be accumulated and only used when they add up
;    to whole steps
;  ---------------------------------------------------------------------
      x_accumulator = xs
      x_increment = (xt - xs) / FLOAT(Idelta_y)

      IF (yt LT ys) THEN BEGIN
        increment = -1
      ENDIF ELSE BEGIN
        increment = 1
      ENDELSE
;  ---------------------------------------------------------------------
;  *********************************************************************
;  find the elevation for every cell between the sensor and the target
;    and store them in elev for the LOS test
;  ---------------------------------------------------------------------

      FOR Iy_cell = Iys, Iyt, increment DO BEGIN
        Ix_cell = FIX(x_accumulator)

;  ---------------------------------------------------------------------
;; for debugging
    IF debug EQ 1 THEN BEGIN
    PRINT, 'Istep, Ix_cell, Iy_cell         =', Istep, Ix_cell, Iy_cell
    ENDIF
;  ---------------------------------------------------------------------

        elev(Istep) = Isntc(Ix_cell,Iy_cell)

;  ---------------------------------------------------------------------
;; for debugging
    IF debug EQ 1 THEN BEGIN
      PRINT, 'elev              =', elev(Istep)
    ENDIF
;  ---------------------------------------------------------------------

        x_accumulator = x_accumulator + x_increment
        Istep = Istep +1
      ENDFOR
    ENDELSE    ; end step in y
;  ---------------------------------------------------------------------
;  *********************************************************************
;  since Istep started at 0, it now gives the number of cells between
;    sensor and target, inclusive
;
;  ---------------------------------------------------------------------
```

```
;; for debugging
  IF debug EQ 1 THEN BEGIN
    PRINT,'Istep               =', Istep
    PRINT, ' '
    PRINT,'----------------EXIT FASTUP-----------------
  ENDIF
; ---------------------------------------------------------------------
; *********************************************************************
;--FROM JANUS(A) 2.0 FANLIN.FOR
; parameters
;   height_s    -sensor height above ground level (feet)
;   height_t    -target height above ground level (feet)
; are added to terrain elevation to give
;   elevation_s -sensor altitude (feet)
;   elevation_t -target altitude (feet)
; ---------------------------------------------------------------------
  elevation_s = height_s + elev(0)
  elevation_t = height_t + elev(Istep - 1)
; ---------------------------------------------------------------------
;; for debugging
  IF debug EQ 1 THEN BEGIN
    PRINT, ' '
    PRINT, 'elevation_s                =',elevation_s
    PRINT, 'elevation_t                =',elevation_t
    PRINT, ' '
  ENDIF
; ---------------------------------------------------------------------
; *********************************************************************
;--FROM JANUS(A) 2.0 QRLOS.FOR
;  LOS_FLAG  (0=NO, 1=YES)
;
; *********************************************************************
;--COMPUTE DELTA-Z ALONG LOS LINE
; there are Istep cells along the path including the endpoints.
;   for LOS testing we only use the interior points.
;
; the sensor and the target are treated as being at the lower
;   valued edge of their cells, so the LOS line starts at the
;      sensor elevation and makes its entire rise or fall in
;        Istep - 1 increments.
; ---------------------------------------------------------------------
;; for debugging
  IF debug EQ 1 THEN BEGIN
    PRINT, ' '
    PRINT, '----COMPUTE DELTA-Z ALONG LOS LINE-------'
    PRINT, ' '
  ENDIF
; ---------------------------------------------------------------------

  Inum_LOS_tests = Istep - 2
  z_increment = (elevation_t-elevation_s) / FLOAT(Istep - 1)
; *********************************************************************
;PRINT, '------QRLOS CALCULATION----------------'
; ---------------------------------------------------------------------
;; for debugging
  IF debug EQ 1 THEN BEGIN
    PRINT, ' '
    PRINT, 'TARGET CELL            =', Istep
```

```
    PRINT, 'SENSOR HEIGHT          =', elevation_s
    PRINT, 'TARGET HEIGHT          =', elevation_t
    PRINT, 'NUM STEPS              =', Inum_LOS_tests
    PRINT, 'z_increment            =', z_increment
  ENDIF
; ---------------------------------------------------------------------
; *********************************************************************
; for every interior point between the sensor and the target, calculate
;   the elevation of the LOS line and compare it to the terrain elevation
;      start with the elevation at the sensor and add z_increment for
;        each step
;
; ---------------------------------------------------------------------
  LOS_line = elevation_s

  FOR I = 1, Inum_LOS_tests DO BEGIN
    LOS_line  = LOS_line + z_increment
    elevation = elev(I)
; ---------------------------------------------------------------------
;; for debugging
  IF debug EQ 1 THEN BEGIN
    PRINT, 'I, LOS_line, elevation  =', I, LOS_line, elevation
  ENDIF
; ---------------------------------------------------------------------
;*********************************************************************
; test to see if LOS exists this far along the LOS line
; ---------------------------------------------------------------------
    IF (LOS_line LT elevation) THEN BEGIN   ;--NO LOS
      RETURN, LOS_FLAG = 0
    ENDIF
  ENDFOR   ; all points have been tested, all OK, LOS EXISTS
; ---------------------------------------------------------------------
;; for debugging
  IF debug EQ 1 THEN BEGIN
    PRINT, '---------LOS EXISTS!!!!!!--------'
  ENDIF
; ---------------------------------------------------------------------
  RETURN, LOS_FLAG = 1
  END
; ---------------------------------------------------------------------
; *********************************************************************
```

FIND_RANGE.PRO:

```
FUNCTION find_range, x0, y0, x1, y1
;
; ************************************************************************
; Bill Kemple                                        last update  15 June 1993
;
;  function to find the range between two points (a sensor and a target
;    perhaps.
;
;  the units are the same as in the passed parameters
;
; ------------------------------------------------------------------------
  xs = FLOAT (x0)
  ys = FLOAT (y0)

  xt = FLOAT (x1)
  yt = FLOAT (y1)

  DIST = SQRT((xt-xs)^2 + (yt-ys)^2)

  RETURN, DIST
  END
; ------------------------------------------------------------------------
; ************************************************************************
```

WEIGHT_BISQUARE.PRO:

```
FUNCTION weight_bisquare, range, max_range, min_weight
;
; ************************************************************************
; Bill Kemple                                    last update  3 august 1993
;
;  bisquare function to find a weight based on the the range
;     between two points.
;
;  the weight starts at w = 1.0 at range = 0.0 and decreases
;     to w = min_weight at range = max_range
;
;  the units for range and max_range should be the same
;
; ------------------------------------------------------------------------
   IF range GE max_range THEN BEGIN
      w = min_weight
   ENDIF ELSE IF range LE 0.0 THEN BEGIN
      w = 1.0
   ENDIF ELSE BEGIN
      mr = float(max_range)
      r  = float(range)
      mw = float(min_weight)
      w = (1.0 - (1.0 - sqrt(mw)) * (r/mr)^2 )^2
   ENDELSE

   RETURN, w
   END
; ------------------------------------------------------------------------
; ************************************************************************
```

WEIGHT_EXPONENTIAL.PRO:

```
FUNCTION weight_exponential, range, max_range, min_weight
;
; ****************************************************************************
; Bill Kemple                                    last update  3 August 1993
;
;  exponential function to find a weight based on the the range
;     between two points.
;
;  the weight starts at w = 1.0 at range = 0.0 and decreases
;     to w = min_weight at range = max_range
;
;  the units for range and max_range should be the same
;
; ----------------------------------------------------------------------------
  IF range GE max_range THEN BEGIN
     w = min_weight
  ENDIF ELSE IF range LE 0.0 THEN BEGIN
     w = 1.0
  ENDIF ELSE BEGIN
     mr = float(max_range)
     r = float(range)
     mw = float(min_weight)
     e_1 = 0.367879
     one_e_1 = 0.632121

     a = (mw - e_1) / one_e_1
     b = (1 - mw) / one_e_1
     w = a + b * exp(-r/mr)
  ENDELSE

  RETURN, w
  END
; ----------------------------------------------------------------------------
; ****************************************************************************
```

WEIGHT_LINEAR.PRO:

```
FUNCTION weight_linear, range, max_range, min_weight
;
; ************************************************************************
; Bill Kemple                                    last update  3 august
1993
;
;  function to find a weight based on the the range
;     between two points.
;
;  the weight starts at w = 1.0 at range = 0.0 and decreases linearly
;     to w = min_weight at range = max_range
;
;  the units for range and max_range should be the same
;
; ------------------------------------------------------------------------
  IF range GE max_range THEN BEGIN
     w = min_weight
  ENDIF ELSE IF range LE 0.0 THEN BEGIN
     w = 1.0
  ENDIF ELSE BEGIN
     mr = float(max_range)
     r  = float(range)
     mw = float(min_weight)
     w = 1.0 - (((1.0-mw) / mr) * r)
  ENDELSE

  RETURN, w
  END
; ------------------------------------------------------------------------
; ************************************************************************
```

ARTY.PRO:

```
PRO ARTY, color
;
; ****************************************************************
; Bill Kemple                          last update: 26 november 1993
;    modified arty01.pro to build arty.pro
;       changed the weight for distance from xy to a trp
;
;    last setup to run 1 nov 93 -- artynew0
;                      2 nov 93 -- artynew30
;
; this procedure determines the distructive potential DP due to
; artillery at each point on the battlefield.
;
; the units are {expected enemy weapons system kills per minute},
; the same as in the direct fire DP surface, so the surfaces can be
; added to give an overall DP surface (less close air support).
;
; factors under the commander's control that determine the surface
; are:
;
;    locations of the artillery firing units (FUs)
;    locations of people who would call fire (FOs)
;    locations of the target reference points (TRPs)
;    locations of the obstacles (OBSTs)
;
; ****************************************************************
;; several print commands have been inserted for debugging. to
;;   invoke them, set debug to 1
;;
;; the function test_LOS can be skipped by setting no_test_LOS = 1
;;
;; the function find_range can be skipped by setting no_find_range = 1
;; ----------------------------------------------------------------
;;  debug gives more output than the buffer can handle, so i'll
;;     use debug2 for finding fos
;;
;;  debug3 is used to test the trafficability routine
 ;
    debug         = 0
    debug1        = 0
    no_test_LOS   = 0
    no_find_range = 0
    debug2        = 0
    debug3        = 0

;; 18 jul 93 - to see how long this guy takes
;;    d1 goes here
;;    d2 goes at the end
;;    diff is the elapsed time
   d1 = today()

; ----------------------------------------------------------------;
; the analysis proceeds as follows:
;
;    the terrain elevation data is brought into the current
```

```
;      workspace from the Isntc file using the RESTORE command.
;
;    since the Isntc entries are just elevations, xy
;     coordinates that correspond to them are generated using a
;       routine copied from putpic.pro
;
;    a real array the same size as Isntc is created to hold the
;       artillery DP surface value at each point, xy.
;
;    data files are read in and arrays and tables are
;       constructed
;
;   the arty DP value is calculated for each point on the
;        battlefield,
;
;   the surface generated by these values is written to 'outfile'.
;       it can be  written as a PostScript file or displayed
;       using the procedure artydraw.pro
;
; ****************************************************************
;
; several user defined functions are used. if the program goes too
; slow, the code can replace the calls. the functions are:
;
;   find_range (x0,y0,x1,y1)
;      determines the range between two points,
;
;   test_LOS (Isntc,x0,y0,height0,x1,y1,height1)
;      adapted from Janus(A)
;      tests for LOS between two points, with the observer at
;         (x0,y0), height0 (feet) above the terrain (Isntc), and the
;           target at (x1,y1), height1 above the terrain.
;
;   trafficability (x,y,s)
;      returns trafficability (s) at (x,y) that may or may not be
;        due to an obstacle.
;    NOTE: no terrain data with trafficability is available at this
;     time, so the trafficability is the min of trafficability_max or
;     the min value assoviated with an obsticle in this cell. The
;     procedure trafficability nis not use at this time.
;
;   weight_bisquare (distance, max_distance, min_weight)
;      a bisquare weight multiplier
;        starts at 1.0 at distance = 0.0,
;          and decreases to min_weight at distance = max_distance,
;
;   weight_linear (distance, max_distance, min_weight)
;      a linear decreasing weight multiplier
;        starts at 1.0 at distance = 0.0,
;          and decreases to min_weight at distance = max_distance,
;
;   weight_exponential (distance, max_distance, min_weight)
;      an exponentially decreasing weight multiplier
;        starts at 1.0 at distance = 0.0,
;          and decreases to min_weight at distance = max_distance.
;
; a description of each function is given in its preamble.
;
```

```
; ****************************************************************
; the first step is to bring the terrain data into the current
;   workspace using the RESTORE command, we are currently
;      using a portion of the NTC in the variable Isntc, which
;        was SAVED into the file 'bill.dat' all Isntc has are the
;          elevations (feet) for each 100m x 100m cell
;
; ----------------------------------------------------------------
  COMMON bill, Isntc, botlftxy

  RESTORE, filename = 'bill.dat'
; ----------------------------------------------------------------
; ****************************************************************
; some initial values that may want to be read in at a later time:
;
; cellsize -
;   the size of a terrain cell.
;   the elevation data contains one number per cell
;   often a cell is 100m x 100m
;   elevation and trafficability are assumed constant throughout a cell
;   coordinates for weapons systems, trps, etc are given in these units
;   e.g. when cellsize = 100,
;         a trp at 551 1200 and a tank at 552 1200 are 100m apart
;
; perfect -
;   the one round fire-for-effect probability of killing an armored
;   vehicle if everything is ideal
;
;   the DP at xy starts out at perfect if any FO has LOS to xy
;   OTHERWISE, THE DP AT XY IS 0.0
;
; fo_max_range -
;   if any FO has LOS to xy, the nearest such is the designated FO.
;   the starting DP (perfect) is reduced by a multiplicative weight that
;   gets smaller as the distance from the designated FO
;   to xy gets greater. this continues until the distance equals
;   fo_max_range. a designated FO farther than this only contributes
;   min_weight
;
; trp_max_range -
;   TRPs only contribute to the DP at xy if there is one within distance
;   trp_max_range of xy that can also be seen by the designated FO. in
;   this case, the TRP multiplier decreases linearly from 1.0 to
;   min_trp_weight
;
; trafficability_max -
;   the open terrain speed of armored vehicles unimpeded by obstacles
;
; min_fo_wt -
;   the smallest multiplier (weight) for an FO that has LOS to the target
;
; min_trp_wt -
;   the multiplier used when nearest TRP the FO has LOS to is more than
;   trp_max_range
;
; min_traff_wt -
;   the multiplier used when the trafficability at xy is greater or equal
;   to trafficability_max
```

```
;
; firingrate - the max sustained rart of fire for the firing units
;    in volleys per minute. for now, 3 at start of battle, 1.5 at end
;    to reflect arty at deminished capacity due to counter battery fires
;; ---------------------------------------------------------------------
    cellsize                = 100.
    perfect                 = 0.9
    fo_max_range            = 6000.0
    trp_max_range           = 2000.0
    trafficability_max      = 35.0
    min_fo_wt               = 0.3
    min_trp_wt              = 0.65
    min_traff_wt            = 0.35
    firingrate              = 3.0

    outfile                 = 'artynew30.srf'
; ---------------------------------------------------------------------
; *********************************************************************
; now we create a floating point array the same size as Isntc to
;    hold the artillery DP surface value at each point, xy.
;
;    since Isntc is an integer array, we use its dimensions explicitly
;       to create a floating point array
;
; ---------------------------------------------------------------------

  surface_size = SIZE(Isntc)
       DPcols = surface_size(1)
       DProws = surface_size(2)

      artysurf = MAKE_ARRAY(DPcols, DProws, /Float)

; ---------------------------------------------------------------------
; *********************************************************************
; next we want to create x and y coordinate values that match up
;     with the elevation data in Isntc
;
;;  copied from putpic.pro.
;     botlftxy is: 45, 99
;       this creates an x_coord vector: 45, 45.1,...,60  (151 values)
;       and a y_coord vector:           99, 99.1,...,109 (101 values)
;
; ---------------------------------------------------------------------
;    x_coord = botlftxy(0)+.1*indgen(151)
;    y_coord = botlftxy(1)+.1*indgen(101)
; ---------------------------------------------------------------------
; *********************************************************************
; the next step is to read in the data and create some arrays
;  or tables.
;
;  our terrain is in 100m cells, so the data in the input files
;     must have 3 digit coordinates (1 unit = 100m)
;
; the data files and the structures are:
;
;  fu.dat
;     gives the type and battery center for each arty btry.
;       read into one dimensional arrays and
```

```
;          stored in FU_TBL
;
;  fo.dat
;     gives the type, location, and height for each observer
;       read into one dimensional arrays and
;       stored in FO_TBL
;
;  trp.dat
;     gives the type, location, and height for each TRP
;       read into one dimensional arrays and
;       stored in TRP_TBL
;
;  obstacle.dat
;     gives the type, location (x and y of lower left corner) and
;       trafficability (speed) for each cell that is in an obstacle
;       read into one dimensional arrays and
;         stored in OBST_TBL
;
;    also, the values in FO_TBL and TRP_TBL are used to compute the
;      array FO_TRP, which gives the LOS information between each FO
;      and each TRP
;
; ****************************************************************
; a similar procedure is used for each of the input files:
;
;    first, create arrays with more rows than will be in the file
;
;    then, read in the data and resize the arrays so they have
;      as many rows as there were in the file.
;
;    next, create arrays for any working variables
;
;    next, build the table - working variables are add to the right
;      as the need for them surfaces.
;
;    finally, get the number of records (rows) in the table
;
; ****************************************************************
;  start with the firing unit info
;
;     fu_reaches_xy is a working variable
; ----------------------------------------------------------------------
   fu_type          = INTARR(20)
   fu_x             = FLTARR(20)
   fu_y             = FLTARR(20)
   fu_max_range     = FLTARR(20)

   status =   DC_READ_FREE('fu.dat', $
              fu_type, fu_x, fu_y, fu_max_range, $
              Resize = [1,2,3,4], /Col)

   fu_int = SIZE(fu_type)
   fu_reaches_xy = MAKE_ARRAY(Size = fu_int)

   FU_TBL = BUILD_TABLE('fu_type, fu_x, fu_y, ' +$
              'fu_max_range, fu_reaches_xy')

   num_FU = N_Elements(FU_TBL)
```

```
; ------------------------------------------------------------------
; col 0 is type
; col 1 is x coord
; col 2 is y coord
; col 3 is max range (meters) for this type arty
; col 4 is reach, a boolian evaluated at each point xy on the
;   battlefield. if this fu can reach xy, it's a 1, ow a 0
; ------------------------------------------------------------------
;; for debugging
  IF debug EQ 1 THEN BEGIN
   FOR i = 0, num_FU -1 DO BEGIN
     PRINT, FU_TBL(i)
   ENDFOR

   PRINT, 'number of firing units, num_FU      ', num_FU

;; print, 'firing units  fu_type, fu_x, fu_y   ', fu_type, fu_x, fu_y
;; print, 'firing units  fu_type, fu_x, fu_y   ', FU
  ENDIF
; ------------------------------------------------------------------
; ******************************************************************
; next, read in the observer info
;   observers are people who would call artillery during a
;     battle. leaders, forward observers, etc. they are
;       identified and listed in a file like fo.dat before this
;         procedure is run.
;
;  fo_index is a working variable used to point back into this
;    table
;
;  fo_rangeto_xy is a working variable. as we evaluate DP at point
;    xy on the battlefield, the distance from xy to fo(i) is
;      temporarily stored there.
;
; fo_sees_xy is a working variable. set to 1 if fo(i) can see xy,
;   set to 0 if he cannot.
;
; ------------------------------------------------------------------
   fo_type          = INTARR(100)
   fo_x             = FLTARR(100)
   fo_y             = FLTARR(100)
   fo_height        = FLTARR(100)

   status = DC_READ_FREE('bluldr_30.dat', fo_type, fo_x, fo_y,
            fo_height, $ Resize = [1,2,3,4], /Col)

   fo_float_size  = SIZE(fo_x)
   fo_int_size    = SIZE(fo_type)

   fo_index      = MAKE_ARRAY(Size = fo_int_size)
   fo_rangeto_xy = MAKE_ARRAY(Size = fo_float_size)
   fo_sees_xy    = MAKE_ARRAY(Size = fo_int_size)


   FO_TBL = BUILD_TABLE('fo_type, fo_x, fo_y, fo_height, ' +$
            'fo_index, fo_rangeto_xy, fo_sees_xy')

   num_FO = N_Elements(FO_TBL)
```

```
    FO_TBL.fo_index      =  INDGEN(num_FO)


;  ------------------------------------------------------------------
;
;  col 0 is type
;  col 1 is x coord
;  col 2 is y coord
;  col 3 is height above terrain for this fo
;  col 4 is range from this fo to xy. reevaluated at each point xy on
;    the battlefield.
;  col 5 reflects LOS from this fo to xy.
;         reevaluated at each point on the battlefield.
;         if this fo can see xy, it's a 1, ow a 0
;
;  ------------------------------------------------------------------
;; for debugging
   IF debug EQ 1 THEN BEGIN
     FOR i = 0, num_FO -1 DO BEGIN
       PRINT, FO_TBL(i)
     ENDFOR

     print, 'number of observers, num_FO      ', num_FO

;;     print, 'observers  fo_type, fo_x, fo_y   ', FO
;;     print, 'observers  fo_type, fo_x, fo_y   ', fo_type, fo_x, fo_y

   ENDIF


;; for debugging
   IF debug1 EQ 1 THEN BEGIN
            PRINT, 'fos read in'
           print, 'number of observers, num_FO      ', num_FO

   ENDIF
;  ------------------------------------------------------------------
;  ******************************************************************
;  read in the target reference point info
;
;  ------------------------------------------------------------------
    trp_type   = INTARR(100)
    trp_x        = FLTARR(100)
    trp_y        = FLTARR(100)
    trp_height = INTARR(100)

    status = DC_READ_FREE('trp.dat',      $
               trp_type, trp_x, trp_y, trp_height, $
               Resize = [1,2,3,4], /Col)


    trp_int         = SIZE(trp_type)
    trp_float       = SIZE(trp_x)

    trp_index       = MAKE_ARRAY(Size = trp_int)
    trp_rangeto_xy = MAKE_ARRAY(Size = trp_float)
```

```
    TRP_TBL = BUILD_TABLE('trp_type, trp_x, trp_y, ' +$
              'trp_height, trp_index, trp_rangeto_xy')

    num_TRP             = N_Elements(TRP_TBL)

    TRP_TBL.trp_index = INDGEN(num_TRP)

;  ---------------------------------------------------------------------
;; for debugging
   IF debug EQ 1 THEN BEGIN
     FOR i = 0, num_TRP -1 DO BEGIN
       PRINT, TRP_TBL(i)
     ENDFOR
     print, 'number of trps, num_TRP       ', num_TRP
   ENDIF

;;      print, 'trps: type, x, y, height    ',  $
;;                trp_type, trp_x, trp_y, trp_height

;; for debugging
   IF debug1 EQ 1 THEN BEGIN
            PRINT, 'trps read in'
            print, 'number of trps, num_TRP       ', num_TRP

   ENDIF

;  ---------------------------------------------------------------------
;  *********************************************************************
;  read in the obstacle info
;    the x and y values give the lower left corner of a cell
;      obstacle cell size is the same as for elevation
;        the entire cell is assumed to be in the obstacle
;
;  ---------------------------------------------------------------------
    obstacle_type       = INTARR(100)
    obstacle_x          = INTARR(100)
    obstacle_y          = INTARR(100)
    obst_traf           = FLTARR(100)

    status =  DC_READ_FREE('obstacle.dat',              $
              obstacle_type, obstacle_x, obstacle_y,    $
              obst_traf,                                $
              Resize = [1,2,3,4], /Col)

   OBST_TBL = BUILD_TABLE('obstacle_type, obstacle_x, ' +$
            'obstacle_y, obst_traf')

   num_OBST = N_Elements(OBST_TBL)
;  ---------------------------------------------------------------------
;; for debugging
   IF debug EQ 1 THEN BEGIN
     FOR i = 0, num_OBST -1 DO BEGIN
       PRINT, OBST_TBL(i)
     ENDFOR
     print, 'number of obstacles, num_OBST       ', num_OBST
   ENDIF
```

```
;; for debugging
  IF debug1 EQ 1 THEN BEGIN
            PRINT, 'obst read in'
            print, 'number of obstacles, num_OBST      ', num_OBST
  ENDIF

; ---------------------------------------------------------------------
;;    num_OBST = N_Elements(obstacle_x)
;;print, 'obs: type, x, y, s  ', obstacle_type, obstacle_x, $
;;            obstacle_y, obst_traf
; ---------------------------------------------------------------------
; *********************************************************************
; next we want to create a FO x TRP matrix (2-d array FO_TRP).
;
;   there is one row for each FO and one column for each TRP.
;      if trp(j) can be seen by fo(i), the (j,i) entry is 1
;      if not, the entry is a 0.
;
; create the array, establish loop bounds, and go
;
; ---------------------------------------------------------------------
  FO_TRP = INTARR( num_TRP, num_FO)

  fo_loop  = num_FO - 1
  trp_loop = num_TRP - 1

    FOR i = 0, fo_loop DO BEGIN
      FOR j = 0, trp_loop DO BEGIN
; ---------------------------------------------------------------------
;; for debugging
  IF no_test_LOS EQ 1 THEN BEGIN
            FO_TRP(j,i)  = 1
  ENDIF ELSE BEGIN
            FO_TRP(j,i)  = test_LOS(FO_TBL(i).fo_x, FO_TBL(i).fo_y,  $
                     FO_TBL(i).fo_height, TRP_TBL(j).trp_x,          $
                     TRP_TBL(j).trp_y, TRP_TBL(j).trp_height)
  ENDELSE

       ENDFOR  ; end of trp loop  (j)
    ENDFOR    ; end of fo loop   (i)
; ---------------------------------------------------------------------
;; for debugging
  IF debug EQ 1 THEN BEGIN
    FOR i = 0, fo_loop DO BEGIN
      FOR j = 0, trp_loop DO BEGIN
        PRINT, i, j, FO_TRP(j,i)
      ENDFOR
    ENDFOR
  ENDIF
;;

  IF debug1 EQ 1 THEN BEGIN
            PRINT, 'fo_trp table done'
  ENDIF


; ---------------------------------------------------------------------
; *********************************************************************
```

```
;  now we walk across the terrain, grid-by-grid. at each cell, we check
;     to see which firing units can hit us.
;
; if any can, we see which FOs have LOS to us. the closest is the
;   designated FO for this xy.
;
; the closest TRP to xy that the designated FO can see is determined,
;   if it is within trp_max_range of xy, there is a TRP contribution
;   if not, min_trp_wt is used.

;
;; 19 jul -- to speed things up
;; FOR x = 465, 470 DO BEGIN
;;     FOR y = 1030, 1035 DO BEGIN
; ------------------------------------------------------------------

  FOR x = 450, 600 DO BEGIN
     FOR y = 990, 1090 DO BEGIN
; ------------------------------------------------------------------
;
;; for debugging
;  IF debug1 EQ 1 THEN BEGIN
           PRINT, 'starting terrain walk'
           print, 'x =   ', x
           print, 'y =   ', y
;  ENDIF

;;*************************************************************** ;;
   the routine to determine which;;, if any, FUs can reach xy, and
;;     multiplies their firing rate;;s into DP at xy, goes here
;;
;; ;; for now, we assume
;; ;;   all FUs can reach all xy
;;     that Pmax for all arty types is .9
  FU_TBL.fu_reaches_xy =  FU_TBL.fu_reaches_xy * 0
  FU_TBL.fu_reaches_xy =  FU_TBL.fu_reaches_xy + 1

   num_FU_that_reach_xy = TOTAL(FU_TBL.fu_reaches_xy)

   E_kills_max = num_FU_that_reach_xy *  firingrate * .9
; ------------------------------------------------------------------
;; for debugging
  IF debug2 EQ 1 THEN BEGIN
     PRINT, 'E_kills_max =  ',E_kills_max
  ENDIF
;
; ------------------------------------------------------------------
; ****************************************************************
; 1) find the range from xy to each FO
; 2) sort them
; 3) starting with the closest, test LOS
; 4) if any FOs have it,
;       a. nearest with LOS is the designated FO.
;       b. his row number in FO_TBL is read from the first element
;           of the index column of the new table "close_FOs"
;       c. the distance from xy to the nearest TRP that the designated FO
;           can see is determined
;       d. the trafficability is determined
```

```
;       e. DPxy is calculated
; 5) otherwise, DPxy = 0.
; ---------------------------------------------------------------------
; 1) find the range from xy to each FO ...

    FOR i = 0, num_FO -1 DO BEGIN
      fox = FO_TBL(i).fo_x
      foy = FO_TBL(i).fo_y
    FO_TBL(i).fo_rangeto_xy = find_range(fox, foy, x, y) * cellsize
    ENDFOR
; ---------------------------------------------------------------------
;; for debugging
  IF debug2 EQ 1 THEN BEGIN
    FOR i = 0, N_Elements(FO_TBL) -1 DO BEGIN
      PRINT, FO_TBL(i).fo_rangeto_xy
    ENDFOR
  ENDIF
; ---------------------------------------------------------------------
; 2) sort them ...

   close_FOs = QUERY_TABLE(FO_TBL, $
  ' * Order By fo_rangeto_xy')
   num_close_FOs =  N_Elements(close_FOs)
; ---------------------------------------------------------------------
;; for debugging
; (2)...
  s = SIZE(close_FOs)
  IF s(0) EQ 0 THEN BEGIN          ; no FOs
     PRINT, 'for some reason, no FOs in sorted table'
  ENDIF                      ;


; ---------------------------------------------------------------------
;;      for debugging
       IF debug2 EQ 1 THEN BEGIN
         PRINT, 'testing LOS, FOs to xy'
         FOR i = 0, N_Elements(close_FOs) -1 DO BEGIN
           PRINT, close_FOs(i)
         ENDFOR
       ENDIF
; ---------------------------------------------------------------------
; 3) starting with the closest, test LOS ...

   i = 0
   desig_FO_flag = 0
   WHILE (desig_FO_flag EQ 0) AND (i LE num_close_FOs -1) DO BEGIN
      desig_FO_flag = test_LOS(close_FOs(i).fo_x, close_FOs(i).fo_y, $
                          close_FOs(i).fo_height, x, y, 0.0)

     IF desig_FO_flag EQ 1 THEN       $ ;LOS exists
        designated_FO_row = close_FOs(i).fo_index

          i = i + 1
        ENDWHILE
        i = 0                          ; just in case
```

```
; 4) if any have it  ...
; 4.a) nearest with LOS is the designated FO ...
; 4.b) his row number in FO_TBL is read from the first element
;   of the index column of the new table "close_FOs"


IF desig_FO_flag EQ 1 THEN BEGIN  ;  (4) LOS exists range_fo_xy  =
FO_TBL(designated_FO_row).fo_rangeto_xy

; ************************************************************
;       <<move on to testing TRPs>>
;       <<this endif will be a long way down>>
; 4.c) the TRP weight is determined ...
;    i. determine if there are TRPs close enough to adjust
;        fire at xy
;
;       (i.1) determine the range from xy to each trp
;
;       (i.2) extract the info for any TRP that is close enough to use
;             for adjusting fire at xy, and
;
;       (i.3) make sure there is at least one.
;
;          (i.3.a) check if any can be seen by the designated FO. if any
;                  can, the closest to xy will be identified first.
;                  its range to xy is used to determine the TRP weight
;
;     ii. otherwise, (none close enough, or none can be seen)
;               the range_trp_xy is set to trp_max_range so the
;                TRP weight will be min_trp_wt
; ------------------------------------------------------------------
;         (i.1)  determine the range from xy to each trp ...

         FOR i = 0, num_TRP -1 DO BEGIN
           trpx = TRP_TBL(i).trp_x
           trpy = TRP_TBL(i).trp_y
           TRP_TBL(i).trp_rangeto_xy = find_range(trpx, trpy, x, y) $
                                              * cellsize
         ENDFOR
; ------------------------------------------------------------------
;;          for debugging
           IF debug2 EQ 1 THEN BEGIN
             FOR i = 0, N_Elements(TRP_TBL) -1 DO BEGIN
               PRINT, TRP_TBL(i).trp_rangeto_xy
             ENDFOR
           ENDIF
; ------------------------------------------------------------------;
(i.2) extract the info for any TRP that is close enough ...

         potential_TRPs = QUERY_TABLE(TRP_TBL,                 $
           ' * WHERE trp_rangeto_xy LE trp_max_range '    + $
           ' Order By trp_rangeto_xy ')

; ------------------------------------------------------------------
;;           for debugging
          IF debug2 EQ 1 THEN BEGIN
            PRINT, 'testing range, TRPs to xy'
           FOR i = 0, N_Elements(potential_TRPs) -1 DO BEGIN
```

```
                PRINT, potential_TRPs(i)
              ENDFOR
          ENDIF
; ---------------------------------------------------------------------;
(i.3)     make sure there is at least one ...

         s = SIZE(potential_TRPs)

; ---------------------------------------------------------------------
;;          for debugging
          IF debug2 EQ 1 THEN BEGIN
            PRINT, 's =   ', s
          ENDIF
; ---------------------------------------------------------------------;
         IF s(0) GT 0 THEN BEGIN          ;(i.3) TRPs are close enough

;          (i.3.a)  check if any can be seen by the designated FO ...

           ITRP = 0
           NTRIALS =  N_Elements(potential_TRPs)
           FOUND_TRP_FLAG = 0

           WHILE ((ITRP LT NTRIALS) AND (FOUND_TRP_FLAG EQ 0)) DO BEGIN
             trp_col      = potential_TRPs(ITRP).trp_index
             fo_row       = designated_FO_row
             LOS_fo_trp   = FO_TRP(trp_col,fo_row)

             IF (LOS_fo_trp GT 0) THEN BEGIN      ;fo can see trp
              FOUND_TRP_FLAG = 1
              range_trp_xy = TRP_TBL(trp_col).trp_rangeto_xy
             ENDIF

             ITRP = ITRP + 1

           ENDWHILE   ; a good trp found or all trps have been tried

           ITRP = 0       ; just in case

;; 1 nov 93 above mod'd to use linear wt with trps vs cookie cutter.
;;  below commented
;           IF  FOUND_TRP_FLAG EQ 1 THEN BEGIN      ;fo can see trp
;              TRP_wt = 1.0
;            ENDIF  ELSE BEGIN ;end fo can see trp
;                              ;start trp close enough, but can't be seen
;              TRP_wt = min_trp_wt
;            ENDELSE

;  ii.  otherwise, (none close enough, or none can be seen)
;                range_trp_xy is set to trp_max_range, causing the
;                TRP_weight to be min_trp_wt ...
;
        ENDIF ELSE BEGIN        ; end (i.3) TRPs are close enough
                        ; start no trps work

           range_trp_xy = trp_max_range

        ENDELSE                 ; end no trps work
```

```
; ***************************************************************
;   (4.d) the trafficability at xy is determined. (if there is more than
;         one obstacle in the cell, the smallest trafficability is used)
;
;   (4.e) the DP due to arty at xy is calculated
;
; ---------------------------------------------------------------------
;;           for debugging
          IF debug3 EQ 1 THEN BEGIN
            PRINT, 'x = ',x, ' y = ',y
            PRINT,'getting ready to query obst_tbl'
          ENDIF
; ---------------------------------------------------------------------
;  20 sep 93... according to pv-wave, the query_table function in this
;     version of wave has bugs and is probably the cause of my
;        problems, so i'm going to code around it for obstacles
;
;               temp_OBST = QUERY_TABLE(OBST_TBL, $
;                  ' * WHERE (obstacle_x EQ x) AND (obstacle_y EQ y)')
; ---------------------------------------------------------------------
;; new           for debugging
;           IF debug3 EQ 1 THEN BEGIN
;             PRINT, 'one'
;           ENDIF
;
;         IF (x GE 468) AND (y GE 1032) THEN BEGIN
;             s = SIZE(temp_OBST)
;                PRINT, 'size temp_OBST =  ',s(0)
;          ENDIF
;
; ---------------------------------------------------------------------;
;               s = SIZE(temp_OBST)
;               IF s(0) EQ 0 THEN BEGIN          ; no obstacles
;
; ---------------------------------------------------------------------
;; new           for debugging
;           IF debug3 EQ 1 THEN BEGIN
;             PRINT, 'two'
;           ENDIF
; ---------------------------------------------------------------------;
;                 traf_xy = trafficability_max
; ---------------------------------------------------------------------
;;           for debugging
;          IF debug3 EQ 1 THEN BEGIN
;            PRINT, 'traf_xy =  ',traf_xy
;          ENDIF
; ---------------------------------------------------------------------
;
;               ENDIF ELSE BEGIN             ;end no obs,
;                                            ;start obstacle(s) exists
;
;               xy_OBST = QUERY_TABLE(temp_OBST, $
;                  ' *  Order By obst_traf')
; ---------------------------------------------------------------------
;
;; new           for debugging
;          IF debug3 EQ 1 THEN BEGIN
;            PRINT, 'three'
```

```
;          ENDIF
; ------------------------------------------------------------------
;
;                  traf_xy = xy_OBST(0).obst_traf
; ------------------------------------------------------------------
;;           for debugging
;           IF debug3 EQ 1 THEN BEGIN
;              PRINT, 'traf_xy =  ',traf_xy
;           ENDIF
; ------------------------------------------------------------------
;
;                ENDELSE                          ;end obstacle(s) exists
;                                                 ;trafficability found
;
; ------------------------------------------------------------------
; ------------------------------------------------------------------
;;  20 sep 93 --- the new stuff starts here

  temp_obst_traf = 0

  FOR i = 0, num_OBST -1 DO BEGIN

      temp_x = OBST_TBL(i).obstacle_x
      temp_y = OBST_TBL(i).obstacle_y

      IF ( temp_x EQ x) AND (temp_y EQ y) THEN BEGIN
          IF  temp_obst_traf EQ 0  THEN BEGIN       ;first obs this xy
              temp_obst_traf = OBST_TBL(i).obst_traf
          ENDIF  ELSE BEGIN                         ;several obs this xy
                                                    ;find worst
         IF OBST_TBL(i).obst_traf LT temp_obst_traf  THEN BEGIN
                 temp_obst_traf = OBST_TBL(i).obst_traf
             ENDIF
          ENDELSE
      ENDIF

  ENDFOR

  IF  temp_obst_traf EQ 0  THEN BEGIN    ; no obs
      traf_xy = trafficability_max
  ENDIF  ELSE BEGIN                      ; obstacle(s) exists
      traf_xy = temp_obst_traf
  ENDELSE
; ------------------------------------------------------------------
; ------------------------------------------------------------------




; 4.e) ...

       artysurf(x-450,y-990) = E_kills_max                          $
       * weight_bisquare(range_fo_xy, fo_max_range, min_fo_wt)     $
       * weight_linear(range_trp_xy, trp_max_range, min_trp_wt)    $
       * weight_exponential(traf_xy, trafficability_max, min_traff_wt)
```

```
; ------------------------------------------------------------------------
;; for debug
     IF debug3 EQ 1 THEN BEGIN
     PRINT, 'x  y  artysurf = ',x,'  ',y,'  ',artysurf(x-450,y-990)
     ENDIF
; ------------------------------------------------------------------------

; 5)...

       ENDIF ELSE BEGIN              ;end (4) LOS exists, start doesn't
          artysurf(x-450, y-990) = 0.0    ;can't shoot here
       ENDELSE                           ;end LOS doesn't  exist

   ENDFOR    ; end of x loop
 ENDFOR      ; end of y loop
; ************************************************************
; write a file that contains the DP surface values.
;     accept the pv-wave default orientation (row)
; ------------------------------------------------------------------------
   status = DC_WRITE_FREE(outfile, artysurf)
; ------------------------------------------------------------------------
   d2 = today()
   diff = dt_duration(d1,d2)
   print, d1
   print, d2
   print, diff

RETURN
END
```

ARTYDRAW.PRO:

```
PRO ARTYDRAW
;
; ****************************************************************
; Bill Kemple
;       last update: 22 nov 1993
;
;       last setup 22 nov 93 -- artynew0.srf --> artynew0.eps
;
; A MODIFICATION TO PUTPIC PRO
; draws either screen images, postscript files for printing,
;    or encapsulated postscript files for inclusion in other
;    software
;
; the color bar on the right is a pain in the ...
;    it has been set to scale itself the same as the surface,
;    but it doesn't behave exactly the same.
;
; ****************************************************************
; we need to get the data in first.
;    this procedure assumes that:
;
;       the terrain has been SAVEd in "terrainfile" and
;          it is already an array named "isntc"
;             of the proper dimensions -- xcols by yrows
;
;       the potential surface is in "surfacefile" and
;          it was written with dc_write_free row oriented (default)
; ****************************************************************
; we are going to put most of the things that are terrain or
;    situation specific here. if this procedure is to be called
;    from another program, we may want them in common or in the
;    call line. but, we can always just edit this before running
;    the program that calls it.
; ----------------------------------------------------------------
  terrainfile = 'bill.dat'
  title = 'Arty Surface at Time=0'
  surfacefile = 'arty0.srf'
  xcols       = 151
  yrows       = 101
  outfile     = 'arty0.eps'

; ****************************************************************
;  if we want to restore variables into a common, declare it here
;
; ----------------------------------------------------------------
   RESTORE, filename = terrainfile

   surf = fltarr(xcols,yrows)

   status=dc_read_free(surfacefile, surf)
; ****************************************************************
; in order to draw axes whose tic marks match the grid
;    coordinates of the terrain, we need to generate coord vectors
;    for example, if the lower left corner,
;       botlftxy is: 45, 99, xcols = 151, and yrows = 101,
```

```
;       this creates an x_coord vector: 45, 45.1,...,60  (151 values)
;       and a y_coord vector:           99, 99.1,...,109 (101 values)
;
; ------------------------------------------------------------------
   x_coord = botlftxy(0)+.1*indgen(xcols)
   y_coord = botlftxy(1)+.1*indgen(yrows)

; ****************************************************************
; the following two lines set the position on the screen of the
;   plot and determine the color table
;
; ----------------------------------------------------------------
   no = 0
;  LOADCT, 5
; ----------------------------------------------------------------
; these two let me bring in a color table that i created with the
;   color_edit function. to use one of the library tables, uncomment
;     the loadct line above and put in the numbe you want
;
  restore, 'bkrgb'
  tvlct, r, g, b
;
; ****************************************************************
; IT SEEMS THAT WE CAN EITHER WRITE A PS FILE OR DRAW A SCREEN
;   IMAGE, BUT NOT BOTH... ONE SET OF THE FOLLOWING NEEDS TO BE
;   COMMENTED OUT
; ****************************************************************
; use the first two lines that follow if you want a postscript file,
;     the first and the third if you want it encapsulated
;----------------------------------------------------------------
  SET_PLOT, 'PS'

;  DEVICE, /Color, Filename = outfile

  DEVICE, /Encapsulated, /Color, Filename = outfile
;
; ****************************************************************
; use the following lines if you want to draw a screen image
;----------------------------------------------------------------;
;  window,no MOD 2,xsize=776,ysize=440,xpos=50+50*(no MOD 2),$
;     ypos=30+300*(no MOD 2)
;
; ****************************************************************
; the following lines draw the colored potential surface
;   -- note that the xrange and yrange are terrain specific
;       we may want to save them with the terrain
;----------------------------------------------------------------

  shade_surf, surf, x_coord, y_coord, az=0, ax=90,            $
     position=[.1,.1,.85,.95], zaxis=-1, shade=bytscl(surf),  $
     xtitle='KiloMeters', ytitle='KiloMeters', /save,         $
     xrange=[45,60], yrange=[99.,109.], ystyle=1

;    zaxis=-1,xtitle='KiloMeters',$
; ****************************************************************
; the following lines draw a contour plot of the terrain
;  again, the xrange and yrange are terrain specific
;     we may want to save them with the terrain
```

```
;-------------------------------------------------------------------
    contour,isntc, x_coord, y_coord, nlevels=25,          $
      position=[.1,.1,.85,.95], /noerase,                 $
      title=title, xrange=[45,60],yrange=[99,109],ystyle=1

;  ***********************************************************
;  the following lines add the control measures to the contour plot
;    they are all terrain and operation specific. we may want to
;    just replace them with text developed in an editor when needed.

;-------------------------------------------------------------------
  x=45+.1*[50,0,18]
  y=99+.1*[0,80,100]
  oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ;phase line plot
  x=45+.1*[80,150]
  y=99+.1*[0,60]
  oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ;phase line plot


  x=45+.1*[72,82,92,98,101]
  y=99+.1*[100,80,80,91,100]
  oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ; piranha

  x=45+.1*[30,35,65,68,40,30]
  y=99+.1*[77,65,63,84,91,77]
  oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ; shark

  x=45+.1*[71,71,102,102,82,71]
  y=99+.1*[60,30,30,40,60,60]
  oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ; cuda

  xyouts,50.2,99.4,'PL Victoria'

  xyouts,53,102.3,'EA Cuda'

  xyouts,49,106.7,'EA Shark'

  xyouts,52.7,108.7,'EA Piranha'
;  ***************************************************************
;  the following lines add a color bar to the right of the plot
;    the colors should match up with the the surface, but it's not
;    perfect

;-------------------------------------------------------------------
  topval = fix(max(surf)) + 1
  color=intarr(2,topval)
  color(0,*)=indgen(topval)
  color(1,*)=indgen(topval)

  shade_surf,color, az=0, ax=90, position=[.88,.1,.93,.95], $
      shades=bytscl(color), xstyle=4,ystyle=4,/noerase

  axis,position=[.88,.1,.93,.95],xstyle=4,                  $
      yaxis=1, yrange = [0,topval], ytitle='Score',/noerase

  DEVICE, /Close_File
```

```
   END
;   ***************************************************************
;   A displayed image can be stored in a Wave variable, and then quickly
;   redisplayed.
;   For a window in a given position, the full picture goes from (0,0) to
;   (x,y) = (max X, max Y) - 1. With the image displayed, it is put into
;   variable image by
;                              image=tvrd(0,0,x,y).
; With a window located at the same place, the image is displayed with
;                               tv,image
;
; Window 0 in putpic is restored with window,no,xpos=50,ypos=30,$
;   xsize=776,ysize=440
; If image captured the previous display, it is then redisplayed with
; above command: tv,image
;
; To save such an image to file, use the save command:
;      save,image,xpos,ypos,xsize,ysize,filename='name',/verbose
;
; It can then be recovered later by restore,'name',/verbose
; ***************************************************************
```

ARTYDRAW2.PRO:

```
PRO ARTYDRAW2
;
; ****************************************************************
; Bill Kemple
;       last update: 26 nov 1993
;
;       last setup 26 nov 93 -- artynew0.srf and artynew90.srf
;                                          --> artynew90.eps
;
;
;; A MODIFICATION TO ARTYDRAW PRO
;
;    reads in both a basecase surface file and the surface file of
;      interest. the basecase file is used to control the colors.
;
; draws either screen images, postscript files for printing
;    or encapsulated postscript files for inclusion in other
;    software
;
; the color bar on the right is a pain in the ...
;    it has been set to scale itself the same as the surface,
;    but it doesn't behave exactly the same.
;
; ****************************************************************
; we need to get the data in first.
;    this procedure assumes that:
;
;       the terrain has been SAVEd in "terrainfile" and
;          it is already an array named "isntc"
;             of the proper dimensions -- xcols by yrows
;
;       the potential surface is in "surfacefile" and
;          it was written with dc_write_free row oriented (default)
;
;       the basecase surface is in "basefile", written the same way
;
; ****************************************************************
; we are going to put most of the things that are terrain or
;    situation specific here. if this procedure is to be called
;    from another program, we may want them in common or in the
;    call line. but, we can always just edit this before running
;    the program that calls it.
;
;     the no= variable is the window number for the surfacefile
;        artydraw puts the base surface in window 0, so use different
;        numbers here for each different window you want to display
;        simultaneously.
; ----------------------------------------------------------------
  terrainfile = 'bill.dat'
  title = 'Arty Surface at Time=90'
  surfacefile = 'artynew90.srf'
  basefile    = 'artynew0.srf'
  xcols       = 151
  yrows       = 101
  outfile     = 'artynew90.eps'
  no = 1
```

```
; ****************************************************************
;  if we want to restore variables into a common, declare it here
;
; ----------------------------------------------------------------
   RESTORE, filename = terrainfile

   surf = fltarr(xcols,yrows)

   base = fltarr(xcols,yrows)

   status=dc_read_free(surfacefile, surf)

   status=dc_read_free(basefile, base)

   topval = fix(max(base)) + 1


; ****************************************************************
; in order to draw axes whose tic marks match the grid
;    coordinates of the terrain, we need to generate coord vectors
;    for example, if the lower left corner,
;       botlftxy is: 45, 99, xcols = 151, and yrows = 101,
;       this creates an x_coord vector: 45, 45.1,...,60  (151 values)
;       and a y_coord vector:           99, 99.1,...,109 (101 values)
;
; ----------------------------------------------------------------------
   x_coord = botlftxy(0)+.1*indgen(xcols)
   y_coord = botlftxy(1)+.1*indgen(yrows)

; ****************************************************************
; the following line determines the color table
;
; ----------------------------------------------------------------
;    LOADCT, 5
; ----------------------------------------------------------------
; these two let me bring in a color table that i created with the
;    color_edit function. to use one of the library tables, uncomment
;       the loadct line above and put in the numbe you want
;
   restore, 'bkrgb'
   tvlct, r, g, b
; ****************************************************************
; IT SEEMS THAT WE CAN EITHER WRITE A PS FILE OR DRAW A SCREEN
;    IMAGE, BUT NOT BOTH... ONE SET OF THE FOLLOWING NEEDS TO BE
;    COMMENTED OUT
; ****************************************************************
; use the first two lines that follow if you want a postscript file,
;      the first and the third if you want it encapsulated
;----------------------------------------------------------------
  SET_PLOT, 'PS'

;  DEVICE, /Color, Filename = outfile

  DEVICE, /Encapsulated, /Color, Filename = outfile

;
; ****************************************************************
; use the following lines if you want to draw a screen image
```

```
;-------------------------------------------------------------
;
;; window,no,xsize=776,ysize=440,xpos=50+50*(no MOD 2),$
;;      ypos=30+300*(no MOD 2)

; ************************************************************
; the following lines draw the colored potential surface
;    -- note that the xrange and yrange are terrain specific
;       we may want to save them with the terrain
;-------------------------------------------------------------

  shade_surf, surf, x_coord, y_coord, az=0, ax=90,            $
     position=[.1,.1,.85,.95], zaxis=-1,                       $
     shade=bytscl(surf,min=0,max=topval),                      $
     xtitle='KiloMeters', ytitle='KiloMeters', /save,          $
     xrange=[45,60], yrange=[99.,109.], ystyle=1

;      zaxis=-1,xtitle='KiloMeters',$
; ************************************************************
; the following lines draw a contour plot of the terrain
;   again, the xrange and yrange are terrain specific
;       we may want to save them with the terrain

;-------------------------------------------------------------
   contour,isntc, x_coord, y_coord, nlevels=25,          $
     position=[.1,.1,.85,.95], /noerase,                  $
     title=title, xrange=[45,60],yrange=[99,109],ystyle=1

;************************************************************
; the following lines add the control measures to the contour plot
;   they are all terrain and operation specific. we may want to
;   just replace them with text developed in an editor when needed.

  ;-----------------------------------------------------------
  x=45+.1*[50,0,18]
  y=99+.1*[0,80,100]
  oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ;phase line plot
  x=45+.1*[80,150]
  y=99+.1*[0,60]
   oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ;phase line plot


  x=45+.1*[72,82,92,98,101]
  y=99+.1*[100,80,80,91,100]
  oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ; piranha

   x=45+.1*[30,35,65,68,40,30]
  y=99+.1*[77,65,63,84,91,77]
  oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ; shark

  x=45+.1*[71,71,102,102,82,71]
  y=99+.1*[60,30,30,40,60,60]
  oplot,x,y,position=[.1,.1,.85,.95],thick=3,/noerase  ; cuda

  xyouts,50.2,99.4,'PL Victoria'

  xyouts,53,102.3,'EA Cuda'
```

```
   xyouts,49,106.7,'EA Shark'

   xyouts,52.7,108.7,'EA Piranha'


;  ****************************************************************
;  the following lines add a color bar to the right of the plot
;    the colors should match up with the the surface, but it's not
;    perfect

;---------------------------------------------------------------------
   color=intarr(2,topval)
   color(0,*)=indgen(topval)
   color(1,*)=indgen(topval)

   shade_surf,color, az=0, ax=90, position=[.88,.1,.93,.95], $
       shades=bytscl(color), xstyle=4,ystyle=4,/noerase

   axis,position=[.88,.1,.93,.95],xstyle=4,                    $
       yaxis=1, yrange = [0,topval], ytitle='Score',/noerase

    DEVICE, /Close_File

END
;  ****************************************************************
;  A displayed image can be stored in a Wave variable, and then quickly
;  redisplayed.
;  For a window in a given position, the full picture goes from (0,0) to
;  (x,y) = (max X, max Y) - 1. With the image displayed, it is put into
;  variable image by
;                                    image=tvrd(0,0,x,y).
;  With a window located at the same place, the image is displayed with
;                                     tv,image
;
;  Window 0 in putpic is restored with window,no,xpos=50,ypos=30,$
;    xsize=776,ysize=440
;  If image captured the previous display, it is then redisplayed with
;  above command: tv,image
;
;  To save such an image to file, use the save command:
;      save,image,xpos,ypos,xsize,ysize,filename='name',/verbose
;
;  It can then be recovered later by restore,'name',/verbose
;  ****************************************************************
```

INITIAL DISTRIBUTION LIST

1. Research Office (Code 08) 1
   Naval Postgraduate School
   Monterey, CA 93943-5000

2. Dudley Knox Library (Code 52) 2
   Naval Postgraduate School
   Monterey, CA 93943-5002

3. Defense Technical Information Center 2
   Cameron Station
   Alexandria, VA 22314

4. Department of Operations Research (Code OR) 1
   Naval Postgraduate School
   Monterey, CA 93943-5000

5. Prof. William Kemple (Code OR/Ke) 2
   Naval Postgraduate School
   Monterey, CA 93943-5000

6. Prof. Harold J. Larson (Code OR/La) 1
   Naval Postgraduate School
   Monterey, CA 93943-5000

7. Director 5
   U. S. Army TRADOC Analysis Command-Monterey
   ATTN: ATRC-RDM
   P. O. Box 8692
   Monterey, CA 93943-0692

8. Director 1
   U. S. Army TRADOC Analysis Command-Monterey
   ATTN: ATRC-RDM (CAPT Fernan)
   P. O. Box 8692
   Monterey, CA 93943-0692

9. Studies and Analysis Division 1
   ATTN: Maj Upton
   MCCDC
   3093 Upshur Ave.
   Quantico, VA 22134-5130

DUDLEY KNOX LIBRARY
3 2768 00337208 7